VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekcia motorových vozidiel v 2D obraze

Detection of Motorized Vehicles in 2D Images

2013 Slavomír Truchlík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Slavomír Truchlík**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Detekce motorových vozidel ve 2D obraze
Detection of Motorized Vehicles in 2D Images

Zásady pro vypracování:

Driver support systems using images are a center of attention recently and various researches on recognizing and understanding the road environment have been done. Road and car detection are fundamental tasks for driver support systems. This work will focus mainly on image processing in the car detection field. The method has to be robust due to the changing road and weather conditions.

1. Become familiar with given relevant areas of image processing, road and car detection area.
2. Implement some of the known approaches used in car segmentation.
3. Try to design, implement and test new method usable in car detection field.
4. Combine different techniques to improve robustness of algorithm in different real life conditions. (dry or wet road, rain, clouds.....)
5. Test the obustness of the algorithm under noisy conditions (different types of additional noises, big differences in illumination...)

Seznam doporučené odborné literatury:

[1] William K. Pratt - Digital image processing 4th edition, Pixelsoft Inc., Los Altos, California, 2007
[2] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins - Digital Image Processing Using MATLAB, Pearson Prentice Hal, December 26, 2003
[3] W. Jones Keeping Cars from Crashing,IEEE Spectrum, vol. 38, no 9, pp. 40-45, 2001.
[4] Zehang Sun, George Bebis, Ronald Miller On-Road Vehicle Detection: A Review. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 28, NO. 5, MAY 2006
[5] Christos Tzomakas, Werner von Seelen Vehicle Detection in Traffic Scenes Using Shadows Ruhr-Universität Bochum, Istitut für Neuroinformatik, Internal Report 98-06, 1998
[6] Chiu, K.-Y. and Lin, S.-F. Lane detection using color-based segmentation. In Proceedings of the IEEE Intelligent Vehicles Symposium. 2005

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Pavel Moravec, Ph.D.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

In Sendai 27. april 2012

.....

I would like to thank Dr. Lukáč, professor Omachi and everyone else who guided me during my research and by that significantly helped with creation of this thesis.

Abstrakt

Počítačové videnie, spracovanie obrazu a strojové učenie patria k najdynamickejšie sa rozvíjajúcim oblastiam modernej informatiky. Ich cieľom je naučiť počítače chápať a rozpoznávať objekty reálneho sveta bez nutnosti zjednodušovania respektíve naučiť počítače vyhodnotiť vstupy reáleneho sveta samostatne a na ich zákalde previesť určitú sériu krokov či opatrení bez explicitného zásahu ľudí. Takýto prístup má široké uplatnenie takmer vo všetkých odvetviach ľudskej práce počínajúc jednoduchným spracovaním obrazu cez automatické rozpoznávanie textu až po samostatné vyhodnocovanie lekárskych snímkov či spresnenie diagnóz. Táto diplomová práca sa zaoberá využitím vyššie spomínaných disciplín v doprave a pomocných navigačných systémoch. Primárnym cieľom výslednej aplikácie je v obrazoch získaných z kamery umiestnenej v prednej časti vozidla rozpoznávať vozovku, ostatné vozidlá a prekážky na ceste čo môže viesť k zvýšeniu bezpečnosti vodičov na cestách a plynulosti premávky.

Klíčová slova: počítačové videnie, spracovanie obrazu, matlab, neurónové siete, Houghova transformácia, strojové učenie, detekcia áut, detekcia vozovky, autonómne systémy

Abstract

Computer vision, image processing and machine learning belong to the most dynamically developing branches of modern computer science. Their goal is to teach computers how to comprehend and distinguish real world objects without any simplifications respectively teach them to evaluate inputs from real world and react individually or execute series of logical steps without explicit interference of people. Such approach has wide use in almost all branches of human work beginning with simple image processing through automatic text recognition ending with automatic evaluation of medical pictures and specifying diagnoses. This diploma thesis focuses mainly on usage of mentioned techniques in traffic adjustment and autonomous driving systems. Primary goal of final application is to recognize road, other vehicles and obstacles on road in images received from camera mounted in the front part of the vehicle which may lead to higher safety and fluency of traffic.

Keywords: computer vision, image processing, matlab, neural networks, Hough transform, machine learning, car detection, road detection, autonomous systems

List of used acronyms and symbols

COLABS	–	Cooperative Laboratory Study Program
RGB	–	Red Green Blue
PGM	–	Pratable Gray Map
CCD	–	Charge-Coupled Device
CMOS	–	Complementary Metal-Oxide Semiconductor
CFA	–	Color Filter Array
HSV	–	Hue Saturation Value
LoG	–	Laplacian of Gaussian
SNR	–	Signal to Noise Ratio
AF	–	Activation Function
BP	–	Back Propagation
ROI	–	Region Of Interest

Contents

List of Tables

List of Figures

1 Introduction

Image processing and machine vision belong without a doubt to the most researched areas in modern computer science. Many companies realize their importance on simplifying and making tasks more effective. Robust and reliable solutions in computer vision can help us to double check results of human workers, make them faster or even completely replace them. Such approach can be very helpful in automation of work and even more importantly in replacing people with autonomous machines in dangerous environments or warn them about dangerous situation before they can realize it them self. Human brain can't work without breaks and is very likely to make mistakes especially when tired or under stress. Imagine a driver on long journey home after hard day in work. He can easily fall asleep and cause traffic accident. Such a situation could be avoided if we had an assistant system that would warn tired driver soon enough that he is in collision course or even adjust the trajectory of vehicle itself. This is just one of many real life usages of computer vision based navigational systems which were my main topic of study in my exchange study period in Japan as COLABS special research student. Before I came to Japan I was mainly working with database and internet related technologies and my knowledge of machine learning and computer vision was very basic so I was afraid that my skills won't be sufficient for research. Fortunately this turned out not to be true and I was able to obtain wide variety of different computer vision based knowledge which I used to finish my work and I could also eventually use them in my future work. Learning this branch of computer science was very interesting mainly because it is very different from conventional ways of programming. More than on technologies it is based on advanced algorithms and computational models.

2 Computer graphics theory

In this section I would like to briefly introduce necessary theoretical background that I was using and might be helpful for readers who are not familiar with discussed topics.

2.1 Image and its representation

If we want to understand digital images it is good to realize where pictures originate in real life. It is the light that carries all image information in form of electromagnetic wave which can be represented as signal. And real life images are in the end nothing else than just two dimensional signal that is processed by our eyes and reflected to our retina. Digital images work in very similar fashion. Image signal goes through lens and afterwards is processed by chip while trying to maintain the maximum of relevant details. While creating digital record we should always know what goal are we trying to achieve and what further post processing is going to be applied on the picture. It is obvious that modeling pictures have different requirements than images created by industrial robot or security camera. Important factor we have to take to consideration is also device that will be used for displaying our pictures. For instance it is unnecessary to create pictures in very high resolution if the only device they will displayed on is small monitor of parking assistant. In our case we are going to present an algorithm which detects road using color as main selecting feature therefore pictures taken in black and white (or shades of gray) would be unusable for our further processing. In computer science we can formally represent picture as continuous function for two variables $f(x, y)$ where f means the brightness value and (x, y) are position indices (although practical implementation is always discrete). We can assume that pictures we are going to work with have finite dimensions $< x_{min}, x_{max} > \times < y_{min}, y_{max} >$. Values of image function can be single number at indices (x, y) in case of monochromatic picture but it can also be set of three (or more) numbers $f_H(x, y), f_S(x, y), f_V(x, y)$ representing value of each component of HSV model. Sometimes it can be even a complex number. In practice values are restricted by technical capabilities of computers. Values of f are after digitalization discrete numbers and indices (x, y) are integers. Whole picture is represented as a matrix

$$A = \begin{vmatrix} \text{f(0, 0)} & \text{f(0, 1)} & \text{...} & \text{f(0, n-1)} \\ \text{f(1, 0)} & \text{f(1,1)} & \text{...} & \\ & & & \\ \text{f(m-1, 0)} & & \text{...} & \text{f(m-1, n-1)} \end{vmatrix}$$

where $x \in < 0, ..., m - 1 >, y \in < 0, ..., n - 1 >$.

2.2 Color spaces

Colors in real world are created by different light wavelengths and human eye is able to recognize them (chromatic light perceived by human sight is approximately in range from 400nm to 700nm). In computer science we are trying to achieve similar goal by specifying a color space or color model representing wavelengths and optical devices that

use them instead of eye. Color model is set of basic colors that can be used for creation of other colors and also defines a set of rules how to combine them to obtain desired result. During years there have been many color models introduced depending on what are we trying to achieve or type of device that will use them. Color spaces are basically a tool that cameras, printers or monitors are using to display or reproduce color. But being able just to create desired color is sometimes not enough, in automatic processing of images we often also need to know the depth or saturation of the color for better understanding of portrayed scene. Color space can be represented as 2D or 3D object showing different attributes of colors and transitions between them.

2.2.1 RGB colorspace

RGB(Red, Green, Blue) model is mostly used for scanning and then showing colors on monitor. It has been standardized in 1931 be CIE commission. According to this model every color is additively composed of three colors - red, green and blue. Technically each color is represented by certain amount of bits (usually it is 8bits for color). By combining basic colors in specific amount we can obtain around 16,7 million colors while using 3x8 bits. In modern graphical software we can find RGBA color model where A stands for alpha channel which represents transparency. RGB can be represented on cube which can be seen in Figure 1.

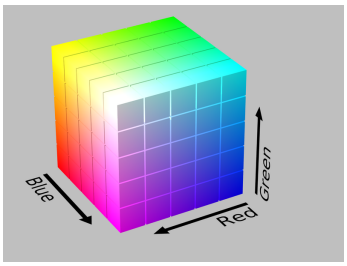


Figure 1: Representation of RGB colorspace as a cube. Adopted from [39]

2.2.2 HSV colorspace

While RGB color space is very convenient to use for displaying colors people don't generally think of colors the same way. It is very hard for human brain do really distinguish nuances of RGB. When we think of colors we tend to group them according some common feature that represents color it self and then make derivatives. For instance we can take a piece of cloth that is of green color. When it's new we will say that color is really vivid but as it's getting older we still think of that color as green but now it's more "washed out". This common color is representing hue component of HSV color space (HSV stands for hue, saturation, value). Formally we define hue as "the property of colors by which they can be perceived as ranging from red through yellow, green, and blue, as determined by the dominant wavelength of light" ([7]). Saturation represents how "warm" the color is respectively how close it is to gray and value shows how bright or

dark color is (if value is at zero color is at all cases black). We represented RGB earlier as a cube with colors defined on axis and limits in corners. HSV color space can be represented as cone that has range of colors on it's perimeter starting with red color on 0°. Other colors are then defined by the angle on the cone respectively how fare are they from 0°. Saturation is represented as distance from center of wide side of the cone. The closer to perimeter color the more saturated it is. Colors directly in the middle have no saturation therefore they are gray. Brightness of color varies by the vertical position in the cone. Colors at the bottom of the cone have no brightness therefore they are pitch black.

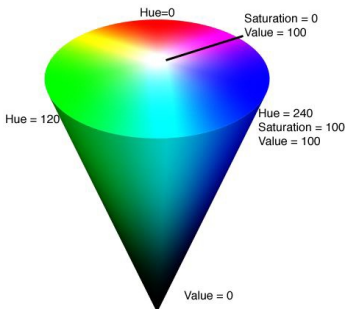


Figure 2: The HSV cone. Adopted from [5]

2.3 Bayer filter and demosaicing

Human eye contains specialized cells that are able to process light. Rods that are not able to distinguish colors and work as sensors for light sensitivity and create monochrome image (there is about 130 million rods in the eye) and cones which are lower in number (only 10 million) and are responsible for color vision. Cones are capable of recognizing only three colors, red, green and blue. Other colors are created in brain as composition of these basic three. Reconstruction of color in digital camera works in very similar way. Chip of every camera is composed of millions of light sensitive cells that are capable of evaluating light intensity (we can think of them as artificial cones). Monochrome image can be created using single charge-coupled device (CCD) or complementary metal-oxide semiconductor (CMOS) sensor which sample the light intensity that is projected onto the sensor and then store this information. Color image can be created in the same way but we need to use three separate sensors and beam splitter for receiving all three primary colors for each pixel. This solution is however very expensive and difficult to construct. Solution for this problem is usage of color filter array (CFA) which allows us to store all three primary colors while using just one sensor. CFA is a filter placed in front sensor that has grid texture made of squares each containing four color cells (light that reaches CFA is filtrated and only specific color component is able to pass through). Generally we are using one red, one blue and two green cells because human eye is most sensitive for green color therefore it is very efficient to have more accurate green color band for reconstructing details. One of the most commonly used CFA was created by Kodak employee

Bryce Bayer which organizes colors as shown in figure 3. This CFA is named after its creator Bayer filter. Bayer color filter replaces the function of rodes in human eye and we can think of it as cameras color sensitive sensor.

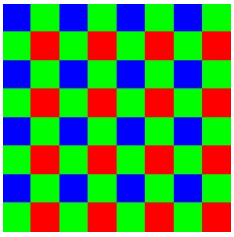


Figure 3: Example of Bayer aligned filter.

2.3.1 PGM file format

Mounted camera that we used for capturing pictures created and stored pictures in PGM format that appears to be in gray scale. Since one of our later algorithms is using color as main feature for road detection it was necessary to convert these pictures into full color. Even though PGM stands for Portable Gray Map it is possible to reconstruct color picture out of informations that are stored in it. This format was created for easy transfer across different systems in plain ASCII format and each pixel is represented as grid of four plain text cells arranged in the same formation as we know from Bayer filter. Process of retrieving full color information out of this alignment is called demosaicing and there are many different algorithms proposed in order to reconstruct as many details as possible.

2.3.2 CFA demosaicing algorithms

Nearest-Neighbor Interpolation is basic and simplest method of interpolation that utilizes 2×2 neighborhood from Bayer aligned pattern (Figure 4) to reconstruct color information. Missing pixel values are simply created from adjacent values as seen in equation 1.

$$B2 = \frac{1}{2} * (B1 + B3) \tag{1}$$

This method unfortunately leads to significant loss of details and color errors especially in edge areas. It can be used for creating previews in environment with limited resources but it is not suitable for real applications.

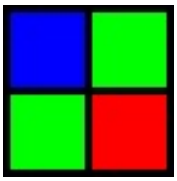


Figure 4: 2×2 Bayer CFA neighborhood. Adopted from [3].

Bilinear Interpolation is also very simple interpolation method that utilizes bigger neighborhood of 3×3 taken from Bayer aligned values and missing colors are computed by averaging nearest pixels as seen in equation 2. Missing red color can be computed the same way as green.

$$B4 = \frac{1}{2} * (B1 + B7) \tag{2}$$

Advantage of this method is that it can be executed using specific convolution kernel. There are two kernels required, one for reconstructing green color and another for reconstruction of red and blue color. Kernels are shown below as F_g for green and F_c for blue and red.

$$Fg = \frac{1}{4} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix} Fc = \frac{1}{4} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

This method provides good results in pictures where color transitions are smooth. However, when sudden change from one to another color appears especially around high frequency areas (mainly edges) result can be poor and significant artifacts occur.

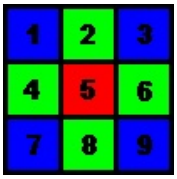


Figure 5: 3×3 Bayer CFA neighborhood. Adopted from [3]

Cubic Interpolation

Works on the same principle as previous algorithm but neighborhood is bigger. In cubic interpolation we are using neighborhood of 7×7 and also the corresponding convolution kernels are much bigger. Cubic interpolation produces the same defects as bilinear interpolation but to a lesser degree. Also occurrence of artifacts is reduced but they can still be seen in final picture. Kernels for cubic interpolation are shown below, again F_g for green color and F_c for blue and red channel.

$$Fg = \frac{1}{256} * \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -9 & 0 & -9 & 0 & 0 \\ 0 & -9 & 0 & 81 & 0 & -9 & 0 \\ 1 & 0 & 81 & 256 & 81 & 0 & 1 \\ 0 & -9 & 0 & 81 & 0 & -9 & 1 \\ 0 & 0 & -9 & 0 & -9 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} Fc = \frac{1}{256} * \begin{bmatrix} 1 & 0 & -9 & -16 & -9 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -9 & 0 & 81 & 144 & 81 & 0 & -9 \\ -16 & 0 & 144 & 256 & 144 & 0 & -16 \\ -9 & 0 & 81 & 144 & 81 & 0 & -9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -9 & -16 & -9 & 0 & 1 \end{bmatrix}$$

High Quality Linear Interpolation is a form of linear interpolation that utilizes inter-channel correlation between different color channels. It uses 5×5 neighborhood in order to improve and make basic linear interpolation more robust. Main difference is that not

only averaged values from neighbor pixels are computed but also corrections obtained from different color channels are applied. Method itself increases computational difficulty just slightly but it reduces amount of artifacts in high frequency areas markedly and can provide even better results than many more complicated non linear interpolations.

Smooth Hue Transition Interpolation

We have described what hue is earlier in section 2.2.2 (HSV colorspace). This interpolation method assumes that hue transitions on the surface of the objects are fluent and color errors occur when sudden change of hue appears across edges. To demosaic Bayer aligned picture we first need to interpolate green channel using previously mentioned methods for linear interpolation and then interpolate the rest of the missing colors. Referring to Figure 5 interpolating of red and blue color is as follows:

$$B2 = \frac{G2}{2} * \left(\frac{B1}{G1} + \frac{B3}{G3}\right) \tag{3}$$

$$B4 = \frac{G4}{2} * \left(\frac{B1}{G1} + \frac{B7}{G7}\right) \tag{4}$$

$$B5 = \frac{G5}{2} * \left(\frac{B1}{G1} + \frac{B3}{G3} + \frac{B7}{G7} + \frac{B9}{G9}\right) \tag{5}$$

Problem of this method is that it fails when value of green color is missing (green is set to zero). As a solution of this problem few simple normalization methods have been introduced.

Gradient-Corrected Bilinear Interpolation

This is the method we have been using to demosaic our pictures because its implementation is used in Matlabs function demosaic(). Our own implementation was not necessary considering the prototype state of algorithms which were oriented on high level segmentation more than accurate low level reconstruction of colors. This approach was introduced in [10] and works as an improvement of algorithm 2.3.2. This method in contrast with standard bilinear interpolation utilizes the fact that gradients among one color can be used to correct errors caused by bilinear interpolation and different color channels are mutually correlated which allows to improve performance. Basic assumption is "that in a luminance/chrominance decomposition, the chrominance components don't vary much across pixels. ([10])". As example we want to interpolate green channel at an R location ($'+'$ marked square in Figure 6). If we want to correct the bilinearly interpolated value $\hat{g}_b(i;j)$ by a measure of the gradient of R at the same location we have to use formula:

$$g_{corrected}(i,j) = \hat{g}_b(i;j) + \alpha\Delta_R \tag{6}$$

where

$$\Delta_R = r(i,j) - r_{avg} \tag{7}$$

and r_{avg} is the average of the 4 nearest red values and α is a gain factor which controls the intensity of such correction. The gradient corrections for blue and red channel are

computed in the same way. Formulas can be found in [10] with additional gain factors β and γ that have been estimated for particular set of images. While maintaining low

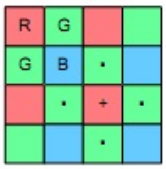


Figure 6: Bayer mosaic of color image. Adopted from [10].

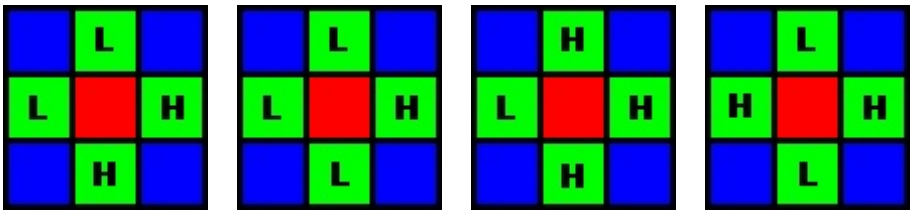
computational requirements and simplicity this method outperforms most of many more complicated nonlienaer methods which makes it the best candidate for us to utilize.

Pattern Recognition Interpolation

This method was proposed by Cok in [4]. According to this method we have to examine green channel around high frequency areas to prevent color mismatches and artifact occurrence. Cok describes how to separate edges into three different groups in green color pane. First we have to examine four neighborhood and compute median value of green. Next by comparing used green pixels to the value of median we are able to evaluate them as high (if value of pixel is greater or equal to obtained averaged value) or low (if value is lower than median). Neighboring pixels then create patterns which we can observe, distinguish and classify according to following rules (cited from [3]):

1. "Pixel is defined as an edge if three neighbor pixels share the same classification."
2. "If two adjacent neighbor pixels have the same classification, then the pixel is a corner."
3. "If two opposite pixels have the same classification, then the pixel is a stripe."

Graphical representation of classification can be seen on Figure 7.Next step is to interpolate missing red and blue values. For this purpose smooth hue transition interpolation described in algorithm 2.3.2 is used. By combining these two methods we are able to create better edge details.



(a) (b) (c) (d)

Figure 7: (7a) is a high edge pattern, (7b) is a low edge pattern, (7c) is a corner pattern and (7d) is a stripe pattern. [pictures adopted from [3]]

3 Image processing theory

With the increase of computational capacity of current computers image processing starts to overlap from theoretical basis into peoples everyday life. We can find examples everywhere around us, image processing techniques are used for compression of pictures, video sequences or editing images published in magazines. Even more advanced techniques can be found more more often in industrial robots, security systems or even in regular low end cameras for making holiday pictures (face recognition, auto adjustment of brightness...). In this chapter I will introduce few common image processing algorithms that I have used in my later work.

3.1 Histogram

Histogram is one of the most basic yet very useful features in image processing. It allows us to examine distribution of brightness across the picture and according to it adjust contrast of unduly exposed pictures. In our case more important function is ability to estimate threshold that will allow us to convert image into so called binary picture (binary picture is created by thresholding). Histogram of digital image with brightness levels in $[0, ..., L-1]$ can be formally defined as a discrete function

$$h(f_k) = n_k \tag{8}$$

where f_k is k-th level of brightness (k $\in$ 0,...,L-1) and n_k is number of pixels in image that brightness level is f_k. In color spaces with more than one color component we can create multiple histograms. For instance in HSV color space we can create three different histograms representing distribution of hue, saturation and value (brightness).

Figure 8: Sample gray scale picture.

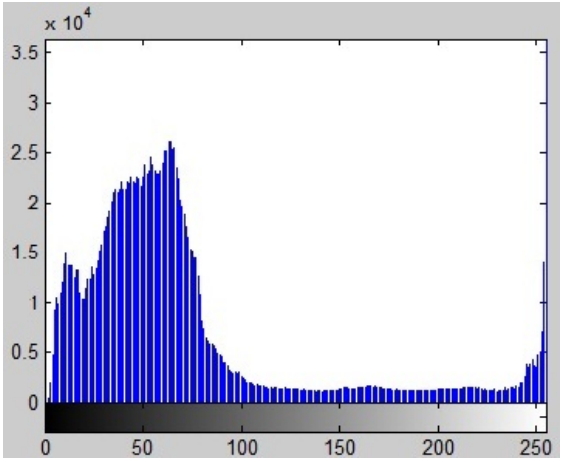


Figure 9: Brightness histogram of Figure 8.

3.2 Thresholding

Thresholding is process of converting images with more brightness levels (gray scale) into barbarized pictures where only 2 different levels exist. The main idea is very simple, first we have to estimate certain threshold which will work as splitting point. All pixels that have brightness higher than this value will be set to 1 (white) and everything else will be set to 0 (black). Which part will be considered foreground and background depends on application respectively what are we trying to achieve. Thresholding can be either global when one threshold is used for whole picture or local where in different parts of image we apply different thresholding values. We can set threshold manually but this approach is not very efficient because lightning conditions in real world may change fast and usage of such a threshold in automatized system would be impossible. Even considering best possible conditions (sunny day) brightness of the same place changes during day time and hard set threshold would lead to overdetection or underdetection in the morning or in the evening. Many techniques of automatic threshold estimation have been proposed (Otsus method of optimal threshold estimation) but they aren't suitable for our case. Instead we will use specialized approach created for car detection problem. Basic assumption is that there are shadows of cars in the picture and those are significantly darker then the rest of the road. According to that we will try to estimate such a threshold that will be able segment only shadow areas and everything else will be considered background and removed. Exact threshold computation will be be discussed later.

Figure 10: Original picture before segmentation.

Figure 11: Thresholded picture.

3.3 Correlation and convolution

Convolution and correlation are filtration methods that take surrounding are of a pixel in cosideration. Convolution is mostly used for edge detection, smoothening and noise reduction. Correlation can be used as similarity scale. Both of this methods are linear. Linear means that if we have operation T, two elements k and l and two constants α and β then then this equation must be valid:

$$T(\alpha k + \beta l) = \alpha T(k) + \beta T(l) \tag{9}$$

Main idea of both methods is a mask (also referred as kernel) that moves across examined area and after each movement we have to multiply values in cells of currently examined

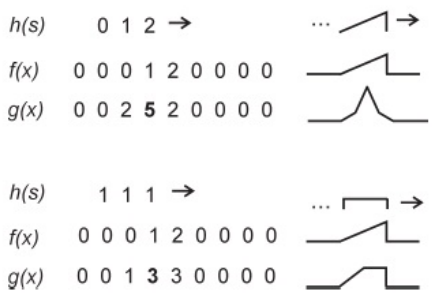


Figure 12: Example of one dimensional correlation (adopted from [6]).

part by values of mask. Results are then added and pixel in the middle of the mask is set to the result value. In case that filtered array and mask are one dimensional we can imagine that mask shifts from left to the right under the array and values that are above each other are multiplied and results added. When filtering image using two dimensional mask we have to take to consideration that in the sides and edges mask overlaps outside of the picture. There are three methods how to deal with this problem:

- Addition of zeros to all sides of picture. Amount of zeros is equal to half size of the mask. This method is called zero padding.
- We won't allow mask to overlap the picture. Picture is then cropped of the borders that have not been filtrated.
- In special cases we can shrink the mask a little.

Which of these approaches will be used depends on application.

3.4 Edge detection

When we take a look at some real world picture human brain thinks of portrayed objects as individual entities that have some common features which allows us to distinguish them from background and organize into sets. For instance when we see picture of on road scene we can easily say what is car, what are people crossing the street or houses. This process of recognizing objects in scene is called segmentation and while for human brain it is very easy in connection with our memory in computer vision it is much more complicated. To make this task a little easier we can think of objects as solid areas surrounded by borders. Process in which are these areas separated from background is called image segmentation. One of the most important techniques of segmentation is border detection which can give as strong clue about where objects are. Borders can be represented as curved lines composed of edges and every edge is created from edge points. Finding whole borders can be very difficult so the usual approach is to firstly detect individual edge points and then connect them into single line. Edge detection is easier to execute in gray scale pictures. Even full color pictures are often first converted to gray scale and afterwards we can look for edge points. Edges are usually located in high

frequency areas which means that edge points are mostly located in places with sudden change of brightness or inflection point. Other approach is to look for whole areas. Since each area is surrounded by solid border we may think that area and border detection should provide identical results. Truth is that in real world images borders can merge with background (yellow t-shirt in from of yellow wall) or can be corrupted by noise. Noise can also lead to over detection and recognition of edges even in places where no objects (light transitions) are located.

3.5 Gradient methods of edge detection

In Figure 13 we can see typical course of brightness through the edge. Gradient methods are based on fact that absolute value of first derivative is high. Value of first derivative describes intensity of contour - size of the edge. Operators that are estimating size of edge are called edge operators. Usually we apply them in every point. As can be seen from these assumptions simplest edge operators would be derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$. Unfortunately they are only able to detect edges parallel with x and y axis. To find edge in general direction we have to examine course of brightness in direction perpendicular to edge. Best clue to decide if there is a edge in particular point is the course in which is the brightness change greatest. This course is gradient and edge itself is perpendicular to gradient. Size of gradient can be also considered size of the edge. Simplest method how

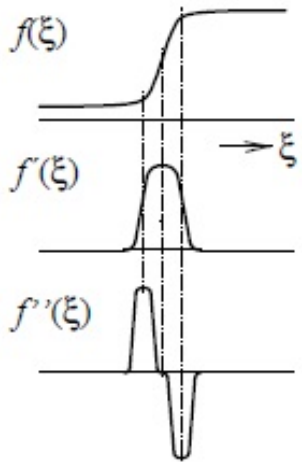


Figure 13: Typical course of brightness and its first and second derivation in edge point.

to decide if point is an edge or not would be by comparing the size of the edge to some pre-set value. This approach has two major disadvantages. First is that edge of object is usually thicker than just one pixel and second we cannot eliminate problems caused by noise. In practice we want generally just to make the decision if some point belongs to the border of an object. So far we have been thinking of image function as continuous but

in practical use we will work with it as discrete. There have been many practical methods for edge detection proposed during the time. Many of them have common feature of gradient computation and often we can represent them as convolution function. Some of the most popular will be described below.

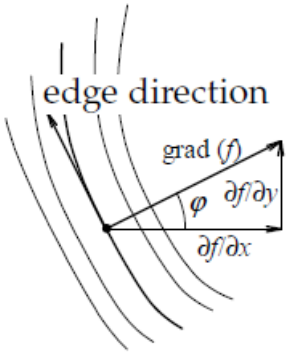


Figure 14: Size and direction estimation using gradient. (adopted from [8])

Roberts operator

One of the first edge operators. It is still used in some cases. Edge size can be estimated by using following convolution mask:

-1	1
-1	1

Table 1: Roberts operator

Same result can be achieved using following formula for point at indices (x, y) where $e(x, y)$ is size of edge and $f(x, y)$ is discrete image function at indices (x, y).

$$e(x,y) = \sqrt{(f(x,y) - f(x+1,y+1))^2 + (f(x+1,y) - f(x,y+1))^2} \tag{10}$$

Prewitt operator

This operator is different from Roberts because for computation are used all the surrounding pixels (that means we are using mask of size 3×3). Operator counts derivatives in directions of both axis by averaging. Let the examined pixels be A, B, C, D, E, F, G, H, I for simplicity. Formula will in that case be:

$$f_x(x,y) = \frac{1}{3} * [(C - A) + (F - D) + (I - G)] \tag{11}$$

$$f_x(x,y) = \frac{1}{3} * [(A - G) + (B - H) + (C - I)] \tag{12}$$

Using averaging in this formula helps to reduce noise in final picture. There are two masks representing Prewitts operator, on for x axis and the other for y axis.

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Table 2: Prewitt operator for x and y axis.

Sobel operator

It is very similar to Prewitt operator, difference is that it uses weighted average for edge size estimation using these formulas:

$$f_x(x,y) = \frac{1}{4} * [(C-A) + 2(F-D) + (I-G)] \tag{13}$$

$$f_x(x,y) = \frac{1}{4} * [(A-G) + 2(B-H) + (C-I)] \tag{14}$$

When using convolution mask coefficients are chosen to emphasize values in the middle of the mask. Gnerally we are able to rotate masks by 45°.

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

0	-1	2
1	0	-1
2	1	0

Table 3: Sobel operator convolution masks rotated by 45°.

Sobel operator proved to be the most effective in finding road lane markings for rough road are estimation and we will use it later for retrieving road samples. Canny edge detector provided also good results but its computational comlplexity was unnecessary and it was slowing down the runtime of algrotihm.

Second derivative and Laplace operator

Figure 13 shows that second derivative of brightness value across the edge has two extremes with opposite sign which changes in the edge point. Since the direction of the edge is generally now known before it is necessary to compute derivation at least in two directions. Second derivative in x direction can be computed as:

$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx [f(x+1,y) - f(x,y) - f(x,y)] - [f(x,y) - f(x-1,y)] \tag{15}$$

after modification

$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx [f(x+1,y) + f(x-1,y) - 2f(x,y)] \tag{16}$$

in analogy we can also create equation for y direction:

$$\frac{\partial^2 f(x,y)}{\partial y^2} \approx [f(1,y+1) + f(x,y-1) - 2f(x,y)] \tag{17}$$

for detection of isolated points we can use Laplace operator which is defined as:

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} = \frac{\partial^2 f(x,y)}{\partial y^2} \tag{18}$$

or

$$\nabla^2 f(x,y) = f_{xx}(x,y) + f_{yy}(x,y) \tag{19}$$

If the image function is discrete we can use differential instead of derivative(can't be used in side lines of image). Differential equations are as follows:

$$f_{xx}(x,y) = f(x-1,y) - 2f(x,y) + f(x+1,y) \tag{20}$$

$$f_{yy}(x,y) = f(x,y-1) - 2f(x,y) + f(x,y+1) \tag{21}$$

And since in digital processing we are working with image function as discrete we can substitute equations 20 and 21 into 19 and obtain final formula for Laplace operator:

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \tag{22}$$

Equation 22 for image function $f(x,y)$ can be easily computed using convolution function mask shown in the first kernel of Table 4. While computing second derivative we can often find also second mask shown in Table 4 which computes derivatives in 0°, 45°, 90°and 135°. Since it is impossible to use these masks for computation in corners and sides we have to deduce special kernels for these cases. Examples are shown in third and fourth Table of 4 (they are showing kernels for second derivation in lowest row of image and left bottom corner, other sides and corners are easily made just by rotating these masks). To make edges more vivid and making picture "sharper" we can add val-

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

0	1	0
1	-3	1
0	0	0

0	1	0
0	-2	1
0	0	0

Table 4: Convolution masks for Laplace operator and example of mask used in sides and corners.

ues of Laplace operator to the original image. This operation is called superposition of image and derivative. This effect can be easily achieved using this mask: Disadvantage

0	-1	0
-1	5	-1
0	-1	0

Table 5: Convolution mask for superposition of image and Laplace operataor.

of Laplace operator is its strong sensitivity to noise. Even small amounts of noise may lead to false edge detection or creation of double edges and it is also impossible to obtain direction of the edge. Nevertheless to this Laplace operator is often used because noise

problem can be efficiently solved in combination with noise reduction respectively image smoothing. After image is smoothed noise doesn't have such a such a huge impact on convolution and resulting image is much clearer.

Method we should combine Laplace operator with is Gauss operator and it was described by Hildreth and Maar in [9]. It is unnecessary to first filtrate picture and then look for edges because both of this techniques can be combined into one called **Gaussian of Laplacian**.

3.5.1 Gaussian of Laplacian

Gaussian of Laplacian is two dimensional function which shape can be seen in picture 15. Advantage of Gaussian is that its "width" can be modulated by adjusting the value of σ. Generally σ varies from 0.4 to 3. Convolution function of Laplacian combined with Gaussian is described by this formula:

$$\begin{aligned}\nabla^2[G(x,y)\times f(x,y)] &= \left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)\iint_{-\infty}^{\infty} G(x-\xi,y-\eta)f(\xi,\eta)\mathrm{d}\xi\mathrm{d}\eta \\ &= \left[\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)G(x,y)\right]\times f(x,y) = \left[\nabla^2 G(x,y)\right]\times f(x,y)\end{aligned} \tag{23}$$

Formula 23 shows that Laplacian of Gaussian can be executed as convolution of image function using function ∇G. If we assume that $r^2 = x^2 + y^2$ then the final convolution kernel will be represented as:

$$\nabla^2 h(r) = -\frac{r^2-2\sigma^2}{2\pi\sigma^6}e^{\frac{r^2}{2\sigma^2}} \tag{24}$$

Convolution mask for this formula is usually defined as 5×5 but occasionally we can use bigger and more complex mask.

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Table 6: Laplacian of Gaussian kernel.

3.5.2 Canny edge detector

I has been proposed and described by J. F. Canny in [12] and [11]. Canny who was running research in low level segmentation decided to create optimal edge detector based on three criteria:

1. **Good detection:** That means that we should be able to find maximum amount of edges while false detection should be reduced to minimum. This corresponds to maximizing the signal-to-noise ratio.

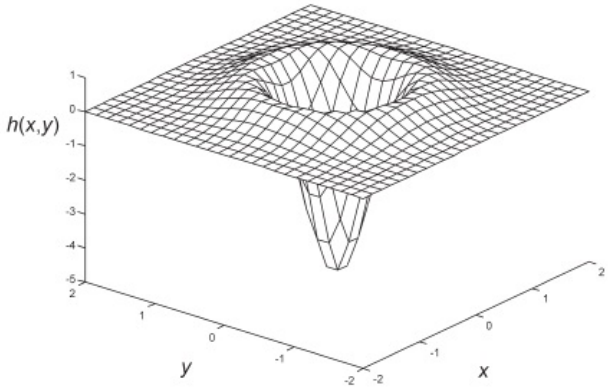


Figure 15: Example of Laplacion of Gaussian function with σ = 0.5 (adopted from [6])

2. **Good localization:** Detected edges should be in ideal case in the center of the real edge.

3. **Only one response to a single edge:** This criterion is implicitly contained in the first one because if we find two different edges for just one border one must be considered false.

   (criteria specification can be found in [12])

Let the impulse response of the filter be $f(x)$, and denote the edge itself by $G(x)$. We will assume that the edge is centered at $x = 0$ and filter to this edge has a finite impulse response bounded by $[-W, W]$. Then the mathematical expression for the first two criteria can be formalized as:

$$SNR = \frac{\left|\int_{-W}^{W} G(-x) f(x) \mathrm{d}x\right|}{\left|n_0 \sqrt{\int_{-W}^{W} f'(x) \mathrm{d}x}\right|} \tag{25}$$

$$Localization = \frac{\left|\int_{-W}^{W} G'(-x) f(x) \mathrm{d}x\right|}{\left|n_0 \sqrt{\int_{-W}^{W} f'^2(x) \mathrm{d}x}\right|} \tag{26}$$

Last thing we want to achieve is to limit the number of peaks in response so that there will be a low probability of declaring more than one edge in high frequency area. Output peaks should be localized in the center of the edge points but additional noise can cause multiple peaks or peak shift. To describe this we are going to use fact that average distance x_{zc} between two zero crossings of response of function G to noise is given by formula:

$$x_{zc} = \pi \left(\frac{-R(0)}{R''(0)}\right)^{(\frac{1}{2})} \tag{27}$$

where $R(\tau)$ is autocorealtion function of function G. Then it's easy to prove that this statement is valid (if we assume that $G(-\infty) = G(\infty) = 0$):

$$R(0) = \int_{-\infty}^{\infty} G^2(x)\mathrm{d}x, R^{''} = \int_{-\infty}^{\infty} G(x)G^{''}(x)dx = \int_{-\infty}^{\infty} G^{'2}(x)\mathrm{d}x \tag{28}$$

since we are looking for average distance x_{zc} between two zero crossings of response of function $f^{'}$ from 27 and 28 we can deduce:

$$x_{zc} = \pi \left(\frac{\int_{-\infty}^{\infty} f^{'2}(x)\mathrm{d}x}{\int_{-\infty}^{\infty} f^{''2}(x)\mathrm{d}x} \right)^{\frac{1}{2}} \tag{29}$$

Distance x_{max} between two adjacent peaks is then x_{zc} doubled. This value is relative to width of operator $< -W, W >$. In that case:

$$x_{max} = 2x_{zc} = kW \tag{30}$$

Ability of the detector to recognize just one ideal edge is the better the lower k is.

Algorithm it self runs in a series of steps that should lead to the best possible result.

1. **Image smoothing:** First logical step is clearing the picture of noise to reduce amount of false edge detections. For this purpose we use basic convolution method and since Gaussian method is effective and maintains low computational resources it is also efficient to use it in Canny detection. General rule is that the bigger the mask is the smaller sensitivity of final detector to noise is. Gaussian kernel used for filtration is shown in Table 7.

2	4	5	4	2
4	9	12	9	4
5	12	15	12	5
4	9	12	9	4
2	4	5	4	2

Table 7: Gaussian kernel for Canny detection with standard deviation of σ = 1.4.

2. **Gradient detection:** After image is cleared of noise we want to search for places with highest light intensity transitions. For this purpose we have to determine gradient of individual points. Gradient evaluation method used in Canny detection is nothing else than Sobel operator described in chapter 3.5. For the best result we have to search for gradient in both x and y directions. (kernels for these operations can be found in Table 3) To estimate magnitude (edge strenght) we compute euclidean distance of G_x and G_y where G_x stands for gradient value in x direction and G_y is gradient value in y direction. Euclidean distance is computed using following formula:

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{31}$$

which is approximately equal to Manhattan distance which is used for lower computational demands.

$$|G| = |G_x| + |G_y| \tag{32}$$

3. **Zero gradient estimation:** Since gradients are perpendicular to the edge it is very efficient to use them for edge direction estimation. However if G_x is equal to zero this estimation fails. Value of zero in x direction means that edge is either parallel to x or y axis depending on G_y (if $G_y = 0$ then also edge direction equals $0°$, if it's $90°$ then edge angle is $90°$). We can avoid this problem by simply ignoring these points but repair code is very simple thus desirable to use:

$$\theta = \arctan(G_y/G_x) \tag{33}$$

4. **Edge direction adjustment:** This step could be described as "clustering" of all possible directions into some reasonable groups. In discrete representation of picture there are only 4 possible directions of surrounding pixels: $0°$(left and right), $45°$(positive diagonal), $90°$(upper and lower pixel) and $135°$(negative diagonal). It is desired then to align various directions to the nearest "normalized" direction. Alignment areas in semicircle are graphically represented in Figure 16. Every angle within certain color area will be rounded to the "middle" value of that zone.

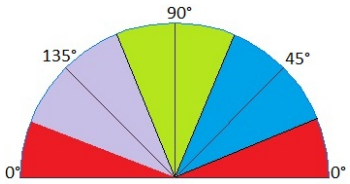


Figure 16: Gradient adjustment areas for edge direction normalization.

5. **Non-maximum suppression:** We can think of this step as thinning of blurred edges into a single sharp line. Idea of this step is to preserve the maximum gradient value and nullify everything else(areas with non-maximum gradient will be set to 0). Algorithm in this steps iterates through edge points and compares edge strength at current pixel with values in both positive and negative gradient direction, this is made easy because values have been already rounded to nearest $45°\theta$ value in previous step(if the gradient direction is $0°$we compare pixels to the left and right). If the edge strength of current pixel is largest we preserve it, otherwise we set it to 0.

6. **Double thresholding:** Edges that are left after non-maximum suppression are probably true edges in the original image but overdetection due to noise can be still present. Since edges are marked by their strength we can simply set a threshold and clear all the edges that are weaker than this threshold. In Canny detection double threshold is used. That means we set two thresholding values and only edges

↑2	↑3	↑5	↑4	↑6
↑4	↑5	↑7	↑6	↑7
↑5	↑6	↑4	↑3	↓2
↑3	↑4	↑3	↓1	↓1

Table 8: Illustration of non-maximum suppression. Edge strengths are symbolized as numbers and color depth while gradient directions is represented by arrows. Resulting edge pixels have white background.

weaker than the lower threshold will be deleted. Rest is marked as a weak edge if gradient value is in between of threshold and strong edge if its gradient is bigger than higher of thresholds.

7. **Hysteresis:** Last step in which we make final decision whether an edge is a real edge or not. Edges that have been marked strong are immediately considered real edges but pixels of weak edges are marked real only if they are connected to a strong edge. Basic logic is that edges created by noise are unlikely to create very sharp light transitions therefore noise caused edges should be only recognized as weak (with well set thresholds) and placed outside of real objects borders.

Wide description of Canny operator has been provided because I am using it as one of the most important segmentation techniques in road detection. Canny operator proved to be very effective in searching for fine details of road lane marking which (in combination with Hough transform) can be later used as very strong clue of where road region is located .

3.6 Hough transform

Picture after edge detection is going to be segmented into a set of separated points but this state does not correspond to native representation of edges. Edge should be continuous area that is strictly separating object from background and another objects. For this reason it is desired to use some technique that is able to connect points into continuous lines representing intact border. In our case we want to scan image for straight lines representing lane markings on the sides and in the middle of the road because they may be a strong lead of where the road region is. Method for detection of straight lines we are going to use is Hough transform which is named after its creator. To detect line in image we are going to use parametric representation of line: $\rho = xcos\varphi + ysin\varphi$. For each point Q exists an infinite amount of lines that are going through it. Each one of these lines can be described by equation $\rho = x_Q cos\varphi + y_Q sin\varphi$. Every line that intersects point Q can be ploted in ρ, φ space as a single point. All lines intersecting this point would then create a graph of function $\rho = x_Q cos\varphi + y_Q sin\varphi$. Taking these premises we can imagine three discrete points A, B and C lying on a single line. Hypothetical lines that can contain these points could be interpreted as sin functions with specific parameters in ρ, φ space. Intersection of these functions would then represent a line which contains all given points. Example can be seen in picture 17.

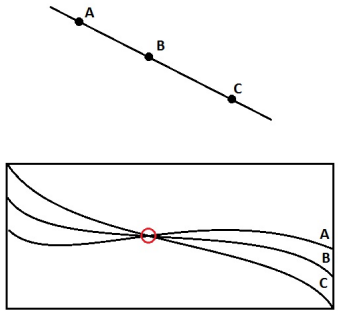


Figure 17: Three points lying at one line and their representation in ρ, φ space.

Given all previous premises we can construct an algorithm that will be able to recognize a line if there is enough of its points detected. For this purpose we have to change ρ, φ space from continuous into discrete and every point i, j in this new space would represent a rectangle for which we can say $\varphi_{i-1} \leq \varphi < \varphi_i, \rho_{j-1} \leq \rho < \rho_j$. Now we create and 2D histogram H in this space and initialize all values with 0. Then we have to iterate through all the detected edge points and for all of them we plot a sin function. All cells of histogram that lie on this function will have their value increased by one. After this every cell $H(i, j) = n$ represents that on a line specified by parameters $\varphi \in < \varphi_{i-1}, \varphi_i >, \rho \in < \rho_{j-1}, \rho_j >$ lies n edge points. Using this approach we can identify lines that contains high amount of points and all values above certain threshold will be considered detected lines. In picture 18 we can see example of Hough detected road lanes. As positive detection we consider only three lines representing left lane (ploted blue), middle line (not highlighted for better transparency) and right lane (plotted yellow). Left and middle lane then create driving region that car is not allowed to leave and left and right lane represent the whole road region which is searched for obstacles.

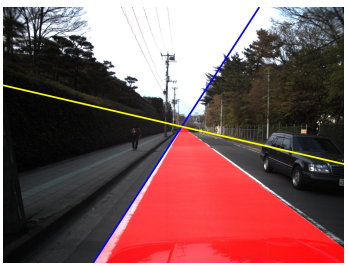


Figure 18: Visualized lane detection output of car detection algorithm.

3.7 Image smoothing

Real-life pictures can be taken under wide variety of conditions which may lead to creation of unwanted elements and artifacts. (noisy pixels can be caused also by bad signal transfer inside of camera) These unwanted elements can be seen as different types

of noises which can lead to major problems in segmentation (for instance creating new edges or destroying real edges which can in our case lead to failure in detection or false positive detection). For this purpose we can use image smoothing algorithms which can to a certain degree surpass noise. There are several methods of noise reduction and they have different results for different type of noises so it's very important to choose appropriate algorithm. Motivation for doing so can be seen in pictures below (all pictures adopted from VŠB - Media Research Lab)

Figure 19: Sample picture without noise.

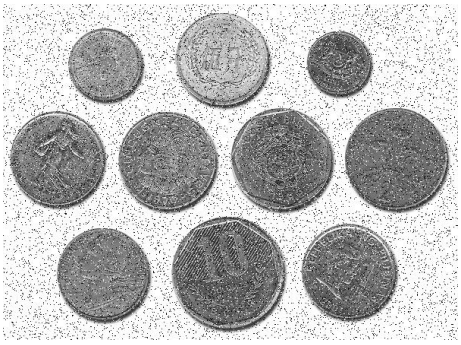


Figure 20: Sample picture with additional noise.

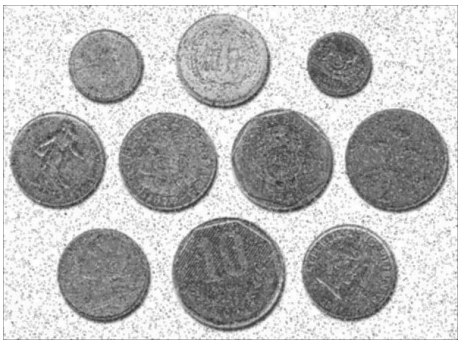


Figure 21: Noisy picture after average filtering.

Figure 22: Noisy picture after median filtering.

Average filtering is one of the simplest of smoothing techniques. It utilizes convolution mask that computes average value of adjacent pixels.

$$g(x,y) = \frac{1}{M} \sum_{s=-smax}^{smax} \sum_{t=-tmax}^{tmax} f(x-s, y-t), \tag{34}$$

where M is number of pixels in mask $M = (2s_{max} + 1) \times (2t_{max} + 1)$. For 3×3 neighborhood we will get

$$g(x,y) = \frac{1}{9} \sum_{s=-1}^{1} \sum_{t=-1}^{1} f(x-s, y-t) \tag{35}$$

and the convolution mask will look as follows:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Table 9: Average filtering kernel.

Median filtering is also very simple smoothing method. Median can be found in sorted array of numbers. If the count of items in array is odd middle value becomes median. If the count is even median is obtained as arithmetical average of two middle numbers. Median filter belongs to group of filters known as "order statistics" or "rank filters". It is important to realize that even though it might look like that from the first look median filtering is not convolution. Principle is shifting of virtual mask across the picture and selecting median values lying underneath this mask. In practice there are also others statistical filter used which as filtering value take general k-th item of an array. General approach while using median filtering would be first to sort the array and then look for middle value. This may take some time because sorting time complexity may vary from $O(Nlog(N))toO(N^2)$ depending on what algorithm we use. When using small mask we can make optimize algorithm by looking for fifth biggest number in case of 3×3 neighborhood and thirteenth biggest value in case of 5×5 neighborhood.

Gaussian smoothing is another filter whose coefficients have different weights. Coefficients closer to the center of the mask are more important in consideration of standard Gaussian curve. One dimensional Gaussian distribution of density is given by formula:

$$h(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-m)^2}{2\sigma^2}} \tag{36}$$

where m is mean value and σ^2 is variance. Value of m corresponds to the center of the mask and σ represents how "steep" the Gaussian function will be. Gaussian varaince with mean value of (0,0) is given by formula:

$$h(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{37}$$

Density function of Gaussian curve is defined in all points from minus infinity to infinity. In discrete situation we have to truncate the mask. Size of the mask is given by parameter σ. All values that exceed certain limit value are considered not important and we don't have to take them into consideration in final result. Examples of one dimensional mask with different values of σ would be:

for σ = 1,5 = [3,28,135,411,800,1000,800,411,135,28,3]

for σ = 3 = [3,11,28,60,135,,249,411,606,800,945,1000,945,800,,606,411,249,135,60,28,11,3]

For high values of σ^2 is final size of mask too big. In case of square mask of size $M \times M$ and picture of size $N \times N$ the time complexity would be $O(M^2N^2)$. For this reason in real applications is we usally utlize separabilty and filtration is executed using one dimensional mask of size $1 \times M$ in x direction and then in y direction.

$$g(x,y) = h(x,y) \times f(x,y) = h(y) \times [h(x) \times f(x,y)]$$

4 Machine learning theory

Machine learning is a field of computer science that is trying to simulate human perception of problems rather than algorithmic approach. Main difference to be considered is while in classical programing we are trying to find most appropriate algorithm that solves certain problem while in machine learning we have set of examples and computer is learning to evaluate these samples by experience (this approach is much more "human-like"). Formally we can define machine learning as "the field of study that gives computers the ability to learn without being explicitly programmed."(Arthur Samuel) This definition is a little older so Tom Mitchell provides a more modern definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." As an example we can use chess.

- E = the experience of playing many games of chess
- T = the task of playing checkers
- P = the probability that the program will win the next game

4.1 Supervised and unsupervised learning

4.1.1 Supervised learning

Dataset of supervised learning could be divided into two groups. First group consists of inputs of our learning problem and second is set or "right" answers for each input. That means instead of looking for correct output (which we already have) we are more looking for the causal relationship between input and output. Once we discover this connection we can use it for estimation of outputs for new inputs that haven't been part of our original set. Supervised learning problems are categorized into **regression** and **classification** problems. In a regression problem we are trying to fit a continuous function through our dataset (that means output for new input will be real number). Classification problems are based on looking for discrete values respectively divide outputs into some finite number of groups.

- Example of regression could be approximate earnings based of shop on its size. Earnings become function of size which is continuous and therefore it is a regression problem.
- Classification problem could be looking for best noise filtration algorithm based on image features (size, frequency spectrum...). As an input we would use set of noisy pictures and as output we would get this set separated into three groups according to what smoothing algorithm is most suitable.

4.1.2 Unsupervised learning

Unsupervised learning is used for problems in which have very limited or even no information about what should results look like. It allows us to find similarities between samples based on relations between variables (features) and that even in case where we are not completely aware of such a bond. Practically we will get a tool that classifies our inputs and divides them into categories based on certain similarity factor. In unsupervised learning there is no teacher that corrects our outputs or leads us to better results based on prediction rules. This method is not only about clustering. It is more about finding associations in sets that may from the first look look very chaotically. Good example of unsupervised learning would be market segmentation when computer would create groups of customers based on similarities between them.

4.2 Liner Regression

4.2.1 Liner Regression with one variable

Linear regression with one variable is basically finding best fitting function of one variable. Input and also output vectors consist of single value so we are trying to find relation between single valued input and exactly one output. This type of linear regression is also known as univariate linear regression. This is example of supervised learning so we know from the beginning what input - output pairs going to look like.

Function that is used to assign outputs to inputs is called **hypothesis**. In this simple case we can use straight line as our hypothesis and that can be represented as linear function $f = ax + b$. By generalizing this formula we can get simplest representation of hypothesis function

$$h_\theta(x) = \theta_0 + \theta_1 x \tag{38}$$

parameters θ_0 and θ_1 can be set to any value. But how are we going to evaluate how good our parameters θ are? Accuracy of hypothesis can be measured using **cost function** which computes average of all outputs from our hypothesis compared to actual outputs(specified by learning set). Simply we could say that cost function computes squared error of differences between results obtained from hypothesis and actual results. Formally is cost function given by formula:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \tag{39}$$

4.2.2 Gradient Descent

Now we have our hypothesis and we have cost function to measure how accurate it is. So how do we automatically find hypothesis that is the best? One approach could be to try all possibilities (brute force) but that wouldn't be very effective. Instead of that we can solve this problem by minimizing cost function. Global minimum of cost function is functional value of our ideal θ parameters. To make this more clear we can imagine that

we are not plotting our hypothesis but cost function. We put θ_0 on the X axis, θ_1 on the Z axis and the cost function on the Y axis. Each point of graph will then represent value of cost function for certain double of parameters. What we are looking for is minimum of such a function (0 in ideal case) which is represented as deepest pit in 2 dimensional case (bottom of convex curve if we only work with 1 dimension). And how do we find such a minimum? We can start at any point in graph (random θ, usually we start at point [0,0]) and then we utilize gradient. If we make derivative (the line tangent to a function) of our cost function we will obtain direction pointing towards minimum (it doesn't have to be necessarily global minimum). And we have to follow this direction by series of steps defined by constant value of α. Using same steps might lead to thought that at certain point we will step over our desired minimum and function will not converge. Gradient descent "guards" it self against this by decreasing size of the step by its nature. The closer we are to the minimum the less steep the curve is. That leads to continuously smaller values of gradient so even if multiply this values by constant α steps are getting smaller closer to the minimum. After we reach minimum gradient is going to be perpendicular to curve so the step will be $0 \times \alpha$ therefore it will stay at minimum even if the iterations continue. The gradient descent algorithm is based on formula:

$$\theta_j := \theta_j - \alpha \frac{\delta}{\delta \Theta_j} J(\theta_0, \theta_1) \tag{40}$$

This formula is repeated until convergence. Generally we can think of it as:

$$\theta_j := \theta_j - \alpha [\text{Slope of tangent - derivative}] \tag{41}$$

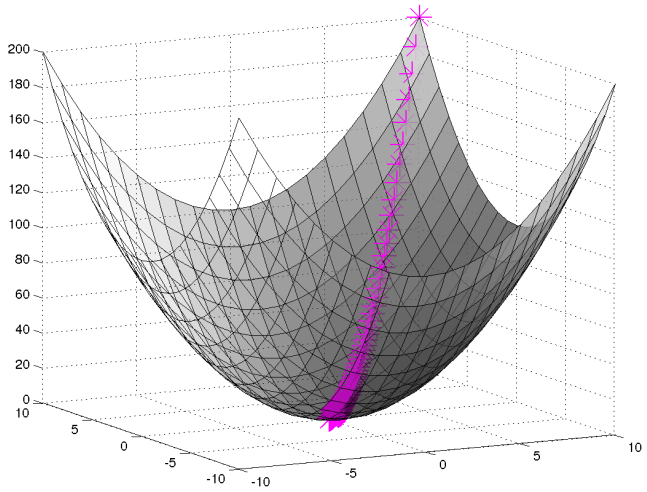


Figure 23: Illustration of gradient descent with continuously smaller steps. Initial point at [0,0] and convergence to minimum.[Adpoted from www.mathworks.com]

If we want to use gradient descent for linear regression we have to make derivatives for each θ. We can substitute our actual cost function and our actual hypothesis function and modify the equation to:

$$\begin{aligned}
&\text{repeat until convergence: } \{ \\
&\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \\
&\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * x^{(i)} \\
&\}
\end{aligned}$$

These formulas are partial derivatives for θ_0 and θ_1. The goal of this approach is that we can start with any hypothesis and after each application of gradient descent formulas we are making our hypothesis more accurate. By the generalization of this algorithm we can create linear regression for any amount of variables (multivariate linear regression). Gradient descent would then look like this

$$\begin{aligned}
&\text{repeat until convergence: } \{ \\
&\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j := 0..n} \\
&\}
\end{aligned}$$

where n is size of vector θ (contains all parameters.)

4.3 Logistic Regression

Logistic regression is used used for classification problems (name might be a little misleading). Purpose of classification is to give discrete output instead of continuous function. In binary classification problem can be our output only 0 or 1. Where 0 is called negative class and 1 positive class. Classification could be theoretically done by normal linear regression by setting threshold to 0.5 and everything below would be negative result and everything above positive result. This approach unfortunately wont work because classification is not a linear function. That's why we are going to use sigmoid function, which will be later utilized also in neural networks. Shape of sigmoid function can be seen in 24 Sigmoid and it is given by formula:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{42}$$

It is logical that when we want to separate something just to class 0 and 1 our hypothesis should satisfy $0 \leq h_\theta(x) \leq 1$ therefore our new hypothesis will be:

$$\begin{aligned}
h_\theta(x) &= g(\theta^T x) \\
g(z) &= \frac{1}{1 + e^{-z}}
\end{aligned} \tag{43}$$

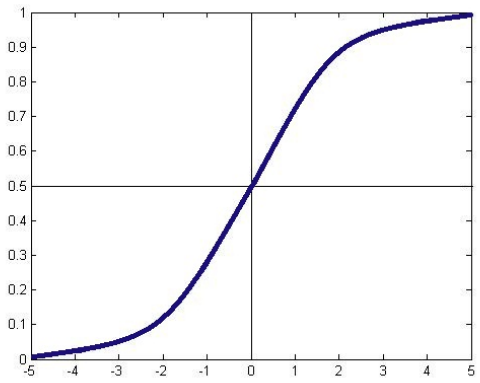


Figure 24: Sigmoid function shape.

This is basically the same hypothesis function as is used in linear regression but it is wrapped in sigmoid g(). The result will be probability that our result is 1. For instance we might get $h_\theta(x) = 0.8$ which means that output of sigmoid will be 1.

4.3.1 Decision Boundary

In order to get our discrete 0 or 1 classification, we can translate the output of the hypothesis function as follows:
$h_\theta(x) \geq 0.5 \rightarrow y = 1$
$h_\theta(x) < 0.5 \rightarrow y = 0$
From graph in Figure 24 we can see that if output is greater than or equal to zero, sigmoids output is greater than or equal to 0.5:
$g(z) \geq 0.5$
$when\ z \geq 0$
So if input to g is $\theta^T X$, then that means:
$h_\theta(x) = g(\theta^T x) \geq 0.5$
$when\ \theta^T x \geq 0$
From these statements we can now say:
$\theta^T x \geq 0 \rightarrow y = 1$
$\theta^T x < 0 \rightarrow y = 0$
The decision boundary is the line that separates the area where y=0 and where y=1. It is created by our hypothesis function.

4.3.2 Cost Function

In classification case it is impossible to use the same cost function as we have been using in case of linear regression. Sigmoid function does not create convex line, instead we will get wavy graph with numerous local minimums so convergence could be very

inaccurate. Cost function for logistic regression will look like this instead:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost((h_\theta(x^{(i)}), y^{(i)})$$
$$Cost((h_\theta(x), y) = -log(h_\theta(x)) \qquad \text{if y = 1}$$
$$Cost((h_\theta(x), y) = -log(1 - h_\theta(x)) \qquad \text{if y = 0}$$

Principle of cost function stays same as it was in linear regression. The further we are away from actual result the higher the penalty for the function will be. And since the outputs are always very close to 0 or 1 correction is always strong, because we are approaching infinity very fast in case of wrong output. If hypothesis output and desired output are the same then cost function is 0.

$$Cost(h_\theta(x), y) = 0 \text{ if } h_\theta(x) = y$$
$$Cost(h_\theta(x), y) \to \infty \text{ if } y = 0 \; and \; h_\theta(x) \to 1$$
$$Cost(h_\theta(x), y) \to \infty \text{ if } y = 1 \; and \; h_\theta(x) \to 0$$

4.3.3 Gradient Descent

Implementation of the gradient descent to obtain minimized cost function is identical with gradient descent of linear regression. Only difference is used cost function which can be written in its final simplified form as:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\, log(h_\theta(x^{(i)})) + (1 - y^{(i)})\, log(1 - h_\theta(x^{(i)})) \tag{44}$$

5 Neural networks theory

Machine learning described in previous chapters works fine until we are not forced to work with non-linear hypothesis. When we want to fit very complex curve we have to use large amount of features (for instance all quadratic and cubic powers of our features) and amount of features is increasing by that very fast. To solve this problem we can use computational model known as neural networks which is based on the same principles as linear and logistic regression (it is still about finding best set of parameters to function, but they are represented a little different way) but offers alternate way of machine learning capable of fitting complex hypothesis with large amount of features. To show why we will use neural networks later we can for example consider a photograph taken in 1600×1200 resolution that was converted into HSV colorspace which has three components. That will make 5 760 000 features for a single picture which makes normal approaches impossible.

5.1 What is neural network?

It is not easy to make an exact definition of neural networks, basically we can say it is a massive parallel processor that simulates experience based problem solving more than algorithmic approach. Neural networks have few major attributes that roughly define them:

- ANN are trying to simulate real biological neural networks that we can find in brain. It is estimated that human brain has $10^{11} - 10^{14}$ neurons and about 10^4 synapses for a singe neuron. It is obvious that real simulation of such a massive parallelism is extremely complicated with our limited resources. Even though that we can't simulate whole brain ANN are trying to follow the same principles and to some degree they should behave the same way (or at least very similar way) as certain parts of brain. The paradox is that even though we can simulate faster processing element - neuron we are not able obtain equivalent results because the final strength of network is based on parallelism.
- Neural networks are making distributed computations. That means that for every computation we are using whole neural network at the same time instead of of just separate parts of memory - memory usage is global. That explains the fact that large amount of neurons is in the end more powerful than few faster simulated neurons.
- All computations in neural networks are based on learning. This is very different approach that the one we are used to. Classic way of solving problems on computers is to find algorithm that is able to process input set and

create new desired output set. This algorithmic phase is completely missing in ANN which in fact works like universal function approximation tool. There are numerous problems that work as functions that we haven't discovered yet and that's when NN come in handy because results we want to obtain will be evaluated according to set of examples we feed to neural network. This set is known as training set and it allows ANN to approximate desired function by experience. That means we are not trying to find general solution of problem but find relation between known instances of this problem.

- In 1949 Dr. Hebb who was Canadian psychologist who was influential in the area of neuropsychology described principles of learning "algorithm" that brain uses to process and save new information in his book The Organization of Behavior. Main idea was that all learning and experience are based on connections between neurons that are adjusted over the time by gaining new knowledge. Knowledge of artificial neural network is stored in the same way respectively everything connections weights between all neurons are adjusted according to result. Weights that led to incorrect answer are weakened while connections that led to expected result are strengthened.

5.2 Model of the neuron

Neurons are the basic elements of ANN. They are based on biological neurons whose simplified model can be seen in Figure 25.

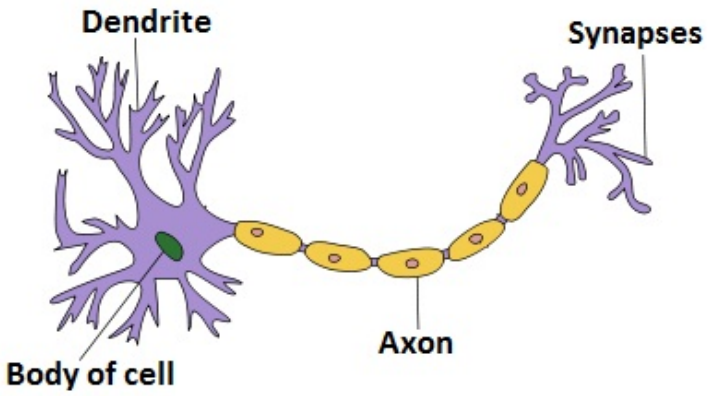


Figure 25: Bilogical model of neuron. Adpoted from [40]

- **Dendrite** represents input to a neuron (place where signal enters).
- **Body of cell** works as computational part. It adds all the signals together and if the final value reaches certain threshold it causes excitation of neuron.

- **Axon** leads signal dependent on excitation.
- **Synapses** are outputs to other neurons and create the NN. Their weights are adjusted during the learning process.

5.2.1 Structure and elemental parts of ANN

Structure of the artificial neuron is very similar to that of biological. Artificial neuron consists of:

1. dendrite - input
2. threshold - value of θ_i which represents output from external world
3. activation function
4. input function o_i
5. synaptic weights - represent synaptic connections (they have direction)

Let's take a closer look to some more complex parts of neuron.

- **Input of neuron** is function (vector) of individual inputs coming from presynaptic neurons. Generally we consider input sum of all partial inputs with respect of weights. Mathematically we can declare it as

$$in_i = \sum_{j=1}^{N} w_{i,j} ou_j + \theta_i \tag{45}$$

  where $w_{i,j}$ are synaptic weights, ou_j are outputs of presynaptic neurons and θ_i is threshold of neuron i. For the first layer of neurons equation can be adjusted like this:

$$in_i = \sum_{j=o}^{N} w_{i,j} ou_j \tag{46}$$

  where $w_{i,j} = \theta_1$ and $ou_j = 1 or -1$. Threshold is basically input from outer world (not output product of other neurons). That means that if there are no presynaptic neurons leading to neuron it accepts only threshold θ_i as input. Neurons like this are called the sigma neurons.

- **Activation function** works as computational unit inside of every neuron. It makes neural network dynamical system that responds to input in_i at the time t. State of neuron i is defined by variable x_i like

$$x_i = f(in_i) \tag{47}$$

  Function $f()$ is then activation function of neuron. There are several activation functions used in neural networks. Most used are:

1. **Linear function** given as $x_i = f(in_i) = in_i$
2. **Signum** which is defined as

$$x = f(in_i) = \begin{cases} 1 & \text{if } in_i \geq 0 \\ 0 & \text{if } in_i < 0 \end{cases}$$

3. **Piecewise-Linear function** This function is defined like this:

$$x = f(in_i) = \begin{cases} 1 & \text{if } in_i \geq \frac{1}{2} \\ in_i & \text{if } in_i \in \left(\frac{-1}{2}, \frac{1}{2}\right) \\ 0 & \text{if } in_i \leq \frac{-1}{2} \end{cases}$$

4. **Sigmoid function** which is most common in ANN, it is defined as

$$x_i = f(in_i) = \frac{1}{1 + e^{-\theta in_i}}$$

where θ is parameter that defines how steep sigmoid will be.

- **Output function** is also very important part of neuron as processing unit. Generally we define it as $ou_i = f(x_i)$ and often it is identical with x_i. Therefore we can say

$$ou_i = O(x_i) = x_i = f(in_i)$$

but we still have to take to consideration that these function don't have to be identical.

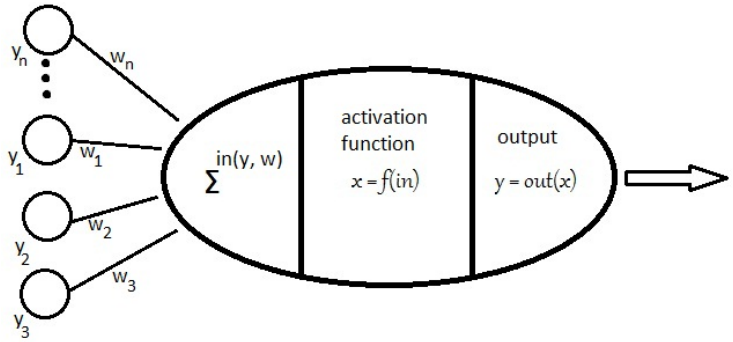


Figure 26: Schematic model of artificial neuron.

5.3 Structure of the neural network

General neural network could be any oriented graph where each vertex (node) represents neuron and oriented edges represent synaptic connections but finding

attributes in these general networks would be very hard. Because of that modern ANN have well defined structure that consists at least of two layers of neurons (input and output) but this kind of network could be used only for linearly separable problems. Because of this usual structure of neural networks also includes one or more hidden layers of neurons. As for connections most commonly we can see that each neuron of $i-th$ layer is connected to all neurons of (i+1-th) layer. Well formed neural network therefore consists of

- **input layer,** this layer only accepts input from outer world, processes them and forwards them into hidden layers.
- **hidden layer** that receives processed inputs (features) and converts them into new set of features which would define the final result. Generally ANN can have n hidden layers but for most cases one hidden layer is enough.
- **output layer** works very similar way as hidden layer but it yields its results to outer world.

According to this division we also separate neurons as input, hidden and output. Topology of neural networks can be divided into two main groups

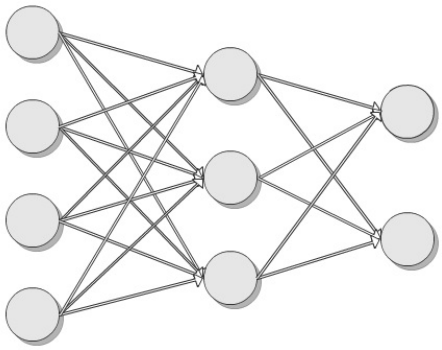


Figure 27: Scheme of feed forward neural network.

- **Feed-forward NN** - information flow can only go one direction from input to output. Example of feed-forward network can be seen in Figure 27.
- **Recurrent NN** - labeling which layer is input, output or hidden in this type of networks is very complicated because some neurons can work as input and output at the same time. Structure of recurrent neural network is shown in Figure 28.

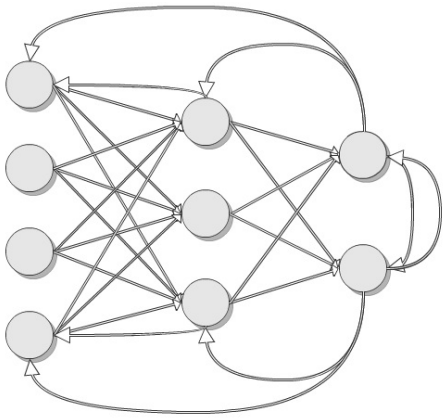


Figure 28: Scheme of recurrent neural network.

5.4 Types of neurons

5.4.1 First generation neuron (McCulloch)

One of the first models that was based on simplified principles. McCulloch created excitatory connections which could only have value of 1 or 0 (having only two classes of connections is mentioned simplification). Every neuron had then specified threshold value and if sum of entering signals was greater than this value neuron was excited. This threshold t was basically very simple activation function which we could define as

$$x = f(in_i) = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} in \geq t \\ 0 & \text{if } \sum_{i=0}^{n} in < t \end{cases}$$

Even this simple structure allows us to realize some operations. For instance we can crate network that works as logical AND operator. AND operator is defined by following truth Table Simple network that recognizes AND function is shown

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Table 10: AND operator truth table.

in Figure 29. It only consists of three neurons, two serve as input and one as output. Alle connection weights between all neurons are 1 and the threshold of output is 1.5. That means that excitation of output neuron can be only achieved if both input neurons fire. As example we can consider value true and false. One true input will send value of one to the output thus it will not fire. If we consider

true and true inputs their added value is 2 which exceeds threshold of output neuron and it will fire (give us result of 1). Similar network could be constructed

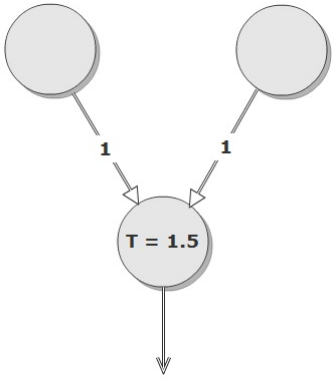


Figure 29: A neural network that recognizes AND logical operation.

also for OR operation. Only difference would be that we have to set the threshold of an output neuron to 0.9.

5.4.2 Perceptron

Perceptron was introduced and described by Rosenblatt in [17] and its primary usage is dichotmic classification - separate input set into two different classes when we assume that these classes are linearly separable. By linearly separable we understand situation when it is possible to divide whole dataset into two different groups using hyperplane. Example of linearly separable set can be seen in Figure 30. Perceptron is one of the most important models used in present

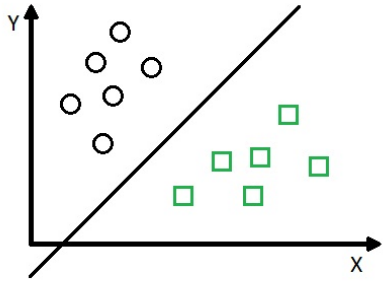


Figure 30: Example of linearly separable example space.

time. Its potential is defined as weighted sum of all input signals. If these signals exceed threshold value of perceptron it becomes excited (emits value 1) otherwise it stays in inhibited state (emits value 0). Let potential of preceptron be ϑ.

Mathematically we can specify this using signum function

$$\begin{aligned} y = Sgn\left(\sum_{i=i}^{n} w_i x_i - \vartheta\right) \\ Sgn(x) = 1x > 0, \\ Sgn(x) = 0x < 0, \end{aligned} \tag{48}$$

by using stable input perceptron with excitation value $x_i = 1$ with respect to neuron $w_0 = -\vartheta$ we can simplify previous equation like this

$$y = Sgn\left(\sum_{i=i}^{n} w_i x_i\right) \tag{49}$$

5.4.3 Continuous perceptron

Whether the neuron will be excited or inhibited is decided by activation function. So far our activation function was a simple threshold that was or wasn't exceeded. For continuous perceptron we use sigmoid function as activation function. Shape of sigmoid can was shown in Figure 24. Mathematical representation of sigmoid function was already introduced but for convenience I am going to list it once more

$$\begin{aligned} y = S(z) = \frac{1}{1 + e^{-\lambda z}} \\ z = \sum_{i=o}^{n} w_i x_i \end{aligned} \tag{50}$$

where S(z) is activation function, z is potential of neuron and λ is coefficient of signum function (defines its shape). From this we can assume following:

- AF only generates values from 0 to 1 where 1 is excitation and 0 is inhibition.
- AF satisfies two asymptotic conditions $t(-\infty) = 0$ and $t(\infty) = 1$.

5.4.4 Adaptation and learning process

Now that we defined neurons and their principles we have to find out how to adjust weights so perceptron is able to correctly separate inputs into desired categories. Learning of perceptron is done using so called Hebbs rule described below:

Assume that (x(1), d^1), (x(2),d^2), ...,(x(m),d^m) is set of examples that will be used for training, x(t) $\in R^n$, $d^t = 1$ if x(t) $\in$ CLS1 and $d^t = 0$ if x(t) $\in$ CLS2.

- Initialize weights and threshold by random numbers (we can not use 0 as we have been using in linear regression).
- If input vector x(t) is correctly classified using w(t) then there is **no change in weights** for next entry (w + 1) therefore
  1. if w(t)x(t) = 1 and x(t) really belongs to CLS1, that means $d^t = 1$ then
  $$w(t+1) = w(t) \tag{51}$$
  2. if w(t)x(t) = 0 and x(t) really belongs to CLS2, that means $d^t = 0$ then we use the same equation as in 51
- if input vector x(t) is incorrectly classified using w(t) then **we have to change weights** for w(t + 1)
  1. if w(t)x(t) = 1 and x(t) really belongs to CLS2, then
  $$w(t+1) = w(t) - \gamma x(t) \tag{52}$$
  2. if w(t)x(t) = 0 and x(t) really belongs to CLS1, then
  $$w(t+1) = w(t) + \gamma x(t) \tag{53}$$

  where γ is learning parameter that can be positive number which satisfies $0 \leq \gamma \leq 1$ and can be either constant or changing during learning time. If we consider Δ a difference between expected and actual value (similar to gradient descent) expressed as
  $$\Delta - d(t) - y(t)$$
  then we can generalize weight adaptation by formula
  $$\Delta = d(t) - y(t)$$
  $$w_i(t+1) = w_i(t) + \gamma \Delta x_i(t)$$

5.5 Multilayer networks and backpropagation rule

Multilayer ANN are state of the art technology used to solve most of machine learning problems. Topology of such a network can be seen in Figure 27. Such a network is created by three and more layers of neurons and all neurons in $i-th$ layer are connected to all neurons of $i+1-th$ layer (full connection of neurons). And how does neural network work? Computing in feedforward network is done in following way:

1. Input layer accepts input vectors from outside world which causes excitation (or inhibition) of input layer.
2. Output signals of first layer are adjusted (strengthened or weakened) by weighted synaptic connections and lead to following layer.
3. Each neuron of first hidden layer sums all input signals and uses them as input parameter for its activation function which may excite it.
4. Same thing is repeated in all hidden layers.
5. Once input signal crosses whole ANN it reaches output layer which emits it outputs to outer world and we can observe them.

Feed forwarding is behavior that was created as a simplification of real neural network which works in very similar way. All inputs that human body takes from outside world are converted into series of signals which are sent to brain and then forwarded through network of neurons for evaluation and segmentation. But simulation of biological network is not everything we want to achieve. What we really want is to have a network that is able to learn and react to new inputs. Other we would have only mechanism that transforms inputs into a set of random outputs which do not really reflect anything. But we don't want our network to only evaluate our samples, we also want it to generalize material provided for learning and react to new inputs using previously gained experience. This is where we cane notice aspects of real artificial intelligence that really approaches human brain.

But how are we going to train the network? How do we change randomly set synaptic connections to ones that will be able to process given problem? Answer is that we will follow the same rules as we have described in classical machine learning problems. Synaptic connections respectively their weights are nothing else than our parameters θ that have to be adjusted to obtain possible hypothesis. Also learning process of ANN is still attempt to minimize cost function that can evaluate how good are results are. To do this we need set of training samples that will be processed by ANN and reaction of NN to these will be then fixed into our network.Formally we define our training set as:

$T = \{\{I_1, O_1\} \quad \{I_2, O_2\} \quad ... \quad \{I_p, O_p\}\}$
$I_i = [i_1, i_2, i_3, ..., i_k], \quad i_j \in < 0, 1 >$
$O_i = [o_1, o_2, o_3, ..., o_l], \quad o_j \in < 0, 1 >$ where

P is amount of training samples
I_i is vector of excitations of input layer composed of k neurons
O_i is vector of excitations of output layer composed of l neurons
i_j, o_j is excitation of $j - th$ neuron of input respectively output layer.

Method that allows us to adjust weights (θ parameters) is called backpropagation and works very similar to gradient descent. We will be again comparing the differences between results we got from hypothesis and expected results to form new set of θ. Before we proceed to mathematical description of backpropagation we can use verbal description as backpropagation intuition

1. We take vector I_i of $i-th$ training sample and use it for excitation of input layer.
2. Feedforward signals to output layer and obtain results of network.
3. Compare expected result given by vector O_i of $i-th$ sample with result obtained from ANN.
4. Compute squared error and send it back to network backwards (starting from output layer to input layer). This error is used to change weights with respect to learning rate (weights that were leading to correct result are strengthened, weights that were leading to wrong result are weakened).
5. After using the whole training set we compute overall error of all training results. If this error is less than expected we can stop, otherwise we repeat whole process once again using same training set (we start new epoch).

Backpropagation (BP) is in the end nothing else than minimizing cost function that can be intuitively written as

$$E = \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{m}(y_j = o_j)_i^2 \tag{54}$$

where

- y_j = obtained (real) output
- o_j = expected result
- p = amount of training examples
- m = number of input neurons

For the exact definition of cost function we have to define few variables

- L = total number of layers
- s_l = number of neurons in layer l
- K = number of output units

full definition of cost function for NN will then be

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)}\ log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)})\ log(1-h_\theta(x^{(i)})_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(\theta_{j,i}^{(l)})^2$$

this term is modified logistic regression. Few nested summation have been added for multiple otuput nodes. In the first part of the equationwe have an additional nested summation that loops through the number of output nodes. In the regularization part we must account for multiple theta matrices. The columns in our current theta matrix is equal to the number of nodes in our current layer. The rows in our current theta matrix is equal to the number of nodes in the next layer. As before with logistic regression, we square every term. New set of weights is obtained by using this formula

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i} + \mu\Delta w_i^{'} \tag{55}$$

where

- η is learning coefficient
- μ is coefficient of importance of previous step weight
- $\Delta w_i^{'}$ is change of weight in previous step

Coefficient μ can take value from 0 to 1. If we set to 0 that means that there is absolutely no connection to previous step therefore weight is changed only according to this step. If we set this parameter to 1 that means that we consider previous step very important and all changes are made with respect to them. Learning parameter on the other hand represents how strong the impact of the change going to be. If we set learning rate high that means that convergence will move very fast but we risk that we will step over our minimum and from certain point the error rate will start to rise. If we set it too low we can expect convergence towards minimum but it might be extremely slow because we will need enormous amount of steps to reach minimum. Partial derivative of error E with respect to weight w_i will show us how is current weight changing the result. If value of derivative is big and positive that means that even the slightest change provided by this weight leads to huge final mistake. That means that value of this weight needs to be lowered. Same approach is used vice versa with negative value of derivative.

Initial state of neural network is that all weights are set to random numbers. This fact is cause of effect when we train more networks with the same straining set we obtain different results. Computation of $\delta w_i^{'}$ is done like this

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \times \frac{\mathrm{d}y}{\mathrm{d}z} \times \frac{\partial z}{\partial w_i} \tag{56}$$

while following is true

$$z = \sum_{i=o}^{n} w_i x_i \tag{57}$$

$$y = \frac{1}{1 + e^{-\lambda z}} \tag{58}$$

$$\frac{\partial z}{\partial w_i} = x_i \tag{59}$$

$$\frac{\mathrm{d}y}{\mathrm{d}z} = y(1 - y)\lambda \tag{60}$$

last thing to resolve is term $\frac{\partial E}{\partial y}$ which can be

$$\frac{\partial E}{\partial y} = (y - o_j)_k \tag{61}$$

in case of input layer and in all other we compute it as

$$\frac{\partial E}{\partial y} = \delta_{ik}.y_j.(1 - y_j)\lambda.xi \tag{62}$$

where sum is first made in all neurons of higher levels. If we want to compute error of current leayer we first have to compute error rate of all neurons in previous layer.

6 Car detection and implementation

Reason why to develop and improve car detection algorithms is mainly improving the safety of drivers and also minimizing damages caused by car crashes. Every year around 10 million people are injured or killed in car related accidents and it is estimated that up to 1-3 percent of the world's gross domestic product is spent in repairing damaged properties and hospital bills that were caused by car accidents [1], [2].

In this chapter I am going to describe overall structure of my practical implementation and algorithms used. Since theoretical background for this algorithms have been described in previous chapters I am going to focus mainly on practical aspects and results that I have obtained during implementation phase.

6.1 Car detection algorithm overview

Car detection is not a simple problem that can be solved intuitively. For human brain it is very simple to distinguish what is happening in the scene we are currently looking at but from digitized image perspective everything we have is just a matrix of numbers that has to be somehow processed. To overcome this problem we have to use approach very common in segmentation and that is focusing on a set of features common for all objects of certain category. For example if we want detect strawberries hidden in green grass we just have to look for the red color that is common for all the strawberries. In car detection this task is more complicated because there is no unified color or shape of the car. Major differences in car appearance are transforming car detection into very challenging problem but there are still some common features that can be used to detect or at least give us intuition of where a car could be. One this features could be symmetry of the left and right side of the vehicle or vertical edges. Exploiting these similarities can be made more effective by adjusting picture so the distinguishable details are more obvious. For instance when we are looking for edges we can transform the picture into birds eye view perspective which strengthens vertical edges and makes them easier to detect.

Feature we are going to focus on in this work is shadow. Shadow is common for all vehicles and it is relatively resistant to perspective. That means that when we are looking for general shape of vehicle that we see from the side classifier we'll obtain will fail if it will have to recognize car we see from front or behind. Shadow on the other hand is always under the car and thus can provide strong clue about car position. Road underneath the car is always darker then the rest of the road even in bad lightning conditions. Taking all this in concern we will get first part of our algorithm - shadow detection. This method will provide regions that very likely contain car but it can also detect lot of noise caused by surrounding objects. (Shadows on buildings, trees, ...)

To avoid the problems with overdetection and make our segmentation more accurate we will proceed to second part of detection - road region estimation. There are many methods how to detect road region in picture that will be introduced later but the main point why to look for it is to create solid region that will be considered area of interest. That means that for every positive detection we will obtain from shadow estimation we will ask if it lies on a road. If not these objects will be automatically considered noise and discarded. Shadows that are within the road region can be considered cars (or generally just obstacles) and marked as final positive detection or used for further examination. Graphical representation of general algorithm can be seen in Figure 31.

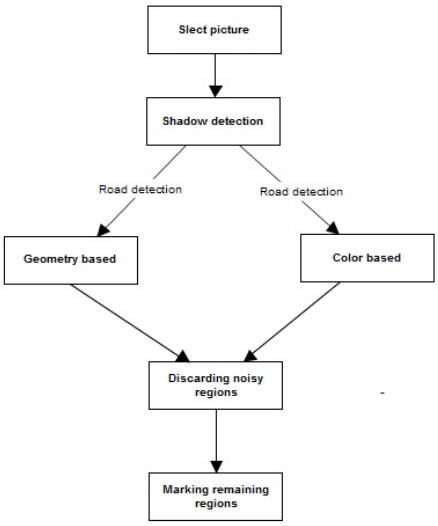


Figure 31: General algorithm flowchart.

6.2 Car candidate region estimation using shadows

6.2.1 Introduction

Computer vision and segmentation problems can be evaluated using many different criteria. Two of the most important are accuracy of detection and time efficiency. Generally we can say that more accurate methods require more examination and therefore they usually have worse time complexity. On the other hand, fast methods can often detect objects we are looking for accurately enough but they also yield lot of false detections. Good solution of this problem is combining several methods and utilizing their advantages. That means we can use fast and dirty method for **initial segmentation** and output of this method will be used as input for more complex verification method which will run faster because it will have to process only small percentage of original picture. The better our

initial guesses are the better the final results will be. In my work I decided to use shadow feature for initial segmentation using algorithm proposed by Tzomakas and Werner in [19].

6.2.2 Detecting shadows

In [20] authors investigated light intensity underneath a vehicle and found out that these areas are distinctly darker than the rest of the pavement road. Since then there were several attempts to distinguish road and shadows however they didn't show how to determine appropriate threshold value for this segmentation. Main problem of threshold estimation is that it heavily depends on the illumination. Illumination is constantly changing during the daytime and can also depend on camera settings. From that it is clear that using fixed threshold will not suffice. Furthermore presence of dark or very bright objects in scene can transform shadow threshold from difficult to almost impossible. In Figure 32 we can see road scenes examined with certain threshold and results we obtained. It is clear that while threshold value 50 is adequate for one picture, it will cause loss of shadows in second one. It is obvious that there is no lower bound for for the car

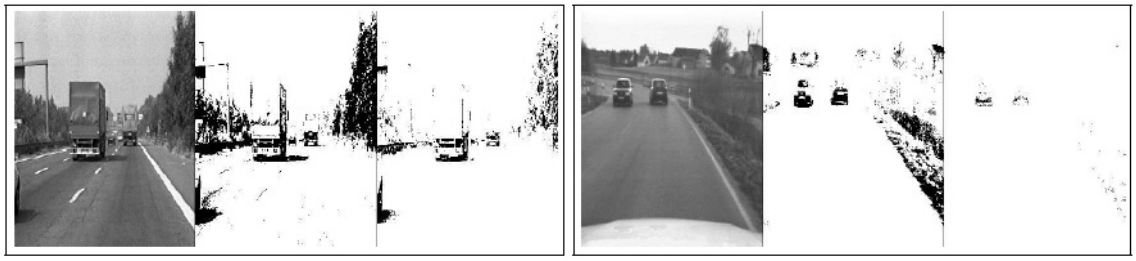


Figure 32: Thresholded images with threshold values t=80, t=50(left) and t=80 and t=60 (right). Adopted from [19]

shadows. But what we can do is define an upper limit for the intensity of the shadow to successfully separate it from its surrounding.

6.2.3 An upper bound for the shadow estimation

Since it is extremely difficult to estimate exact threshold for for an image to optimally separate shadows we will use simplification that is able to provide similar results. This simplification is to determine upper bound of gray value of road and expect the shadows be beneath this level. To obtain good sample of road to examine we could mount another camera pointing to the ground few centimeters in front of a car but this solution would make system more expensive, complex and there is still risk of pointing directly at lane marking which would provide bad results. The ppproach we have used is to look for lowest homogeneous region delimited by edges and consider it road. Texture based approaches provided

more accurate results but they were significantly slower and the the edge based method proved to be adequate enough. Result of road estimation can be seen in figures 33 and 34.

Figure 33: Picture where we want to estimate road region for gray value examination.

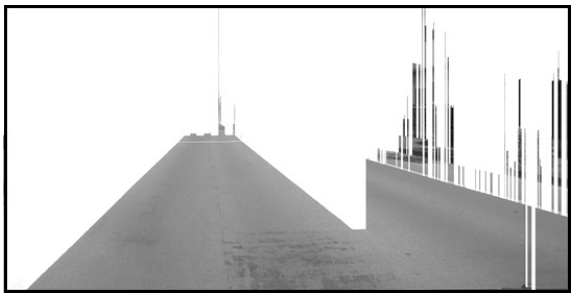


Figure 34: Pictre 33 after edge delimiting.

Assuming normal distribution of the gray values we can fit Gaussian curve to the histogram of the pixels of the road and estimate mean gray value of the road and its variance. If $f(x)$ is the frequency distribution of the road pixels it will be approximated as:

$$f(x) \approx k.e^{\frac{-(x-m)^2}{2\sigma^2}} \tag{63}$$

where m is the average and σ the standard deviation of the curve. Upper bound for the shadows can be defined as the limit where distribution of the road gray values declines to zero (on the left of the mean), that means

$$S_{thresh} = m - 3\sigma \tag{64}$$

This assumption about the road pixel distribution provides good results in most cases even in cloudy weather or when car stands in a larger shadow (for instance shadow of building). Unfortunately, the estimation starts to fail when it becomes darker and light diffusion suppresses the detection of the threshold. Results of shadow detection can be seen below. In these sample pictures we can see noise created by front part of car and surrounding. The front part of car is always the

Figure 35: Sample picture to present shadow detection.

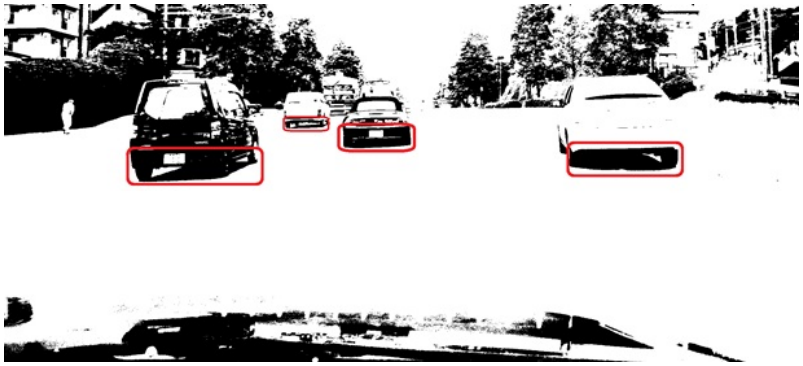


Figure 36: Pictre 35 after shadow estimation.

same size so it can be solved by simply cutting it out of the picture. Methods for reducing amount of surrounding noise will be introduced later in this chapter.

6.2.4 Extracting edges

To finalize our result we have to obtain edges that correspond to the shadows underneath the vehicles. These are usually horizontal direction so our first approach was using normal edge detector in horizontal direction. This solution provided good result but it can be implemented much more efficiently. If we want to find edges we have to simply look for transition between dark and bright areas in vertical direction. These transitions can be easily discovered by shifting the image vertically by a few pixels and subtract from original. Result obtained this way has less detected edges and also significantly reduces amount of surrounding noise. By extracting horizontal edges we reduced amount of information almost to minimum. In fact after the edge detection we have only 2% left of original picture. This result is very good since it provides strong clue about where car regions are and since it was reduced to this low extent it will be very efficient to use it as input for our verification algorithm. To see the amount of final in-

formation I highlighted resulting edges in examined pictures which can be seen below. Results are showing that detection is successful even with the soft part of the shadow or when car stands in shadow of a building.

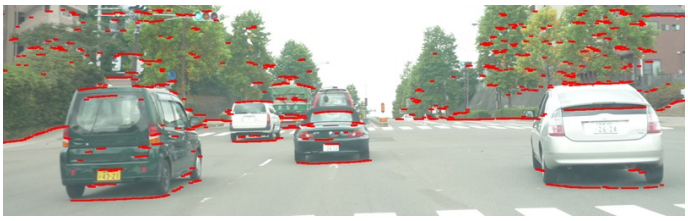


Figure 37: Sample picture with highlighted shadow edges.

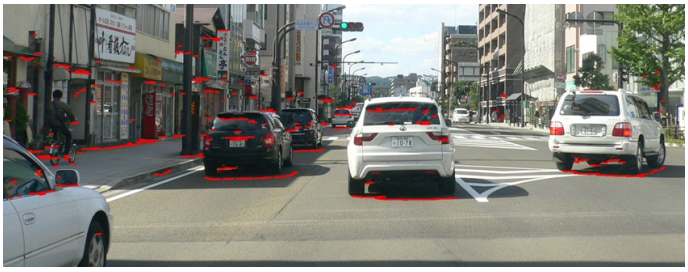


Figure 38: Sample picture with highlighted shadow edges.

6.3 Road detection under a dynamically changing light source

Most of the informations included in this chapter have already been published in [37] that was written along with dr. Lukáč during my internship in Tohoku university. The paper [37] has been written and published as one of the goals of this thesis and therefore I will also use texts and results obtained in this thesis.

Color tracking is a well known technique to distinguish blobs of a single color from the environment and is widely used for tracking as well as for segmentation in low level image processing. In the real world image processing the color tracking is however a difficult problem because the variations in the environmental conditions (lightning, visibility, etc) change the color representation in the color spectrum. One of the problems with color tracking is that the color in the real color spectrum is highly sensitive to the ambient light intensity and color variations. This is true in particular for the roads (streets) where one single road surface can be covered by shadows or can have its color changed based on how the street light is shining. In this chapter we present a machine learning approach to the road (street) surface detection from two-dimensional images, based on the color detection under variable illumination. The proposed method is using an Artificial Neural Network (ANN) that determines specific set of values to isolate

the color spectrum of the desired tracked blob (street) in the HSL color-space: because this spectrum is intensity invariant, each color is situated within a particular region of the Hue/Saturation value range. Using this property the ANN can be taught to efficiently determine such set of color values in the color-space dimension that successfully allow the desired color blob extraction.

6.3.1 Introduction

In machine vision, color detection is a quite important technique [21, 22] used in general in the low level vision algorithms for blob extraction or image segmentation. It is quite often used as a tracking method or as a starting point for object detection; a technique with the goal to determine the position in spatial coordinates of a particular colored blob or of a part of a larger multi-colored object.

One of the main problems of the color tracking approach is the sensitivity to changes in ambient light; tracking a red color under a diffuse white light will not work when the light source is a point light source or if its intensity is halved for instance. Thus tracking a particular color becomes problematic in real-world environments as the light intensity naturally varies due to shadows, occlusion, tunnels or just transient weather states. In addition, different types of lighting will change the colors perceived by the image sensor, such as moving from daylight to fluorescent light. The performance of the various most common color-tracking algorithms was recently reviewed by Sattar and Dudek [25].

The most recent works in the real-time light color tracking include the work of Hayashi and Fujiyoshi [26] or the work of Stern and Efros [27]. Hayashi and Fujiyoshi present an approach where the changes of the illumination are incorporated into the mean-shift tracking algorithm by modifying the color model of the currently observed real-world by estimated luminance of the ambient light [26]. On the other hand Stern and Efros use a color-model switching approach for face tracking; depending on the light intensity the color-model used is changed so as to preserve the the unique color representation of the tracked object (face).

In the area of road extraction various techniques have been applied up to now. Area of road extraction is one of the most important tasks in intelligent vehicles systems. We can find real-life usage of such algorithm beginning with intelligent car systems helping to improve road safety ending with fully automatically guided motorized vehicles. Systems, that allows us automatically adjust speed according to situation, effectively and accurately detect obstacles to prevent collision or keep vehicle on a roadway are in active development for at least last 20 years. Various techniques have been applied up to now utilizing devices such as radar, lidar, electromagnetic waves or GPS. The problem of these approaches is that they often require expensive equipment, therefore we decided to utilize vision based approach based on continually less expensive cameras. Some vision based techniques require a 3D camera to estimate the road based on the

depth information [28], however methods using only single 2D images in general use either color information or geometry information. For instance in [29] was presented road detection system using supervised ANN and Support Vector Machines (SVM) exploiting RGB color-space. In work [30] from Kuo-Yu Chiu and Sheng-Fuu Lin a hybrid model of road estimation combining color based segmentation was described to detect lane marking candidates and then finding road geometry assuming the lanes are continuous lines in vertical direction on which we can use geometrical model like Hough transform. In [31] was presented algorithm that is built on selecting road features and HSV color space, which is introduced as more accurate and effective distinguishing and inhabiting shadows than RGB. In most recent works [32] there are attempts to create robust universal system to determine vanishing point of road which gives a strong clue about its relative position.

The above methods represent computational approach to the color detection/-tracking problem in most cases situated in the RGB color-space. Different colors-spaces could however make the color detection easier because of the different color representation (as illustrated by the approach in [27]). One of the alternatives to the RGB color-model, the HSL model was established as a more *natural* model of the color representation. In particular it separates the color-space in such manner that the natural colors are defined in a 2D plane (Hue and Saturation) and the intensity only modifies these colors between black (minimum intensity) and white (maximum) intensity and thus not altering the essential colors themselves. Thus the HSL model could facilitate the color detection/mechanism if the overall computation requirements could be reduced to the computational requirements of the RGB based color tracking.

In this chapter we propose a system that from a data set learns a set of parameters for estimating the proper color range including mostly the road surface. We present a method that includes a manual sample preparation, automatic pre-processing and automatic learning with verification. The proposed method shows that when combined with a pixelising (density) algorithm the road can be reliably detected with at least as such a good result when done manually by a human user.

6.3.2 Road Surface Detection In The HSL Image Space

The HSL (Hue, Saturation, Luminance/Intensity) color-space is in general seen as a relatively-low computational resources color transform that can be used for low level preprocessing of RGB image data. It was introduced in order to avoid certain disadvantages of the RGB color-space and still remain a low-computational cost transform. As such, one of the main advantages is that the color representation changes in a more meaningful way with changes in ambient lighting conditions than it would in RGB color-space.

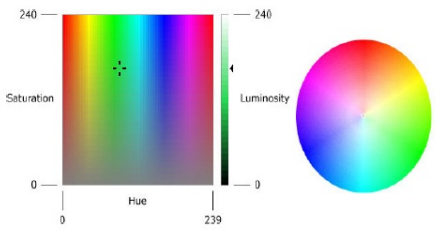


Figure 39: The HSL color-space schematic representation. (a) the representation of all values of H, S and L and (b) color map for fixed Luminosity.

The HSL color-space is shown in Figure 39; Figure 39 (a) shows the HSL color-space in a planar representation with the three dimensions annotated. Figure 39 (b) shows the HSL colors when the Luminosity is fixed. The transformation from RGB to HSL is considered to be low cost and low complexity. For illustration of the RGB to the HSL color-space transform is shown below in Algorithm 1. The complexity of this algorithm can be shown to be of the $O(n)$. In addition, the most complex operation required is division.

Algorithm 1 RGB to HSV transformation

for each pixel in RGB **do**
  r, g, b = color of current pixel
  $v = max(r, g, b)$
  $s = v - min(r, g, b)$
  if $v == 0$ **then**
    $s = 0$
    $h = 0$
  else
    if $r == v$ **then**
      $h = (g - b)/s$
    end if
    if $r == v$ **then**
      $h = 2 + (b - r)/s$
    end if
    if $r == v$ **then**
      $h = 4 + (r - g)/s$
    end if
    $h = h/6$
    if $h < 0$ **then**
      $h = h + 1$
    end if
  end if
end for

Using the HSL color-space, the task of tracking a blob of color lightly changes than in the RGB color-space. In particular, one can detect a given color - a combination of single values of Hue and Saturation for various amount of Intensity (Luminance) by designing a set of threshold values that will allow to isolate the particular color to be tracked.

Unlike in the RGB color-space, where light intensity results in the overall color spectrum difference, in the HSL model the change of the light luminosity significantly changes only one of the values of the color-space. This means that in order to track an object under a light changing source - and assuming that the color of the light is not changing - only one value of the HSL color-space has to be dynamically modified.

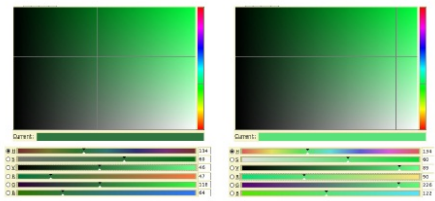


Figure 40: The comparison for a same color under different intensity of light. (a) green color under with stronger light intensity (b) the same color under weaker light intensity.

As an example, Figure 40 shows one selected color with two different light intensities. For comparison, each color is shown both in the HSV and in the RGB color space (sliders below images). The color is given in H and S dimensions with values 124 and 60 respectively. The Figure 40 (a) has V value 46 and Figure 40 (b) has V value of 89. Observe that as shown on the sliders below each of the images when the intensity of the light changes, only one slider V changes value in the HSV (the HSV and the HSL color-spaces are similar color-spaces differing only in details but in general are considered are interchangeable) representation. In the RGB color-space representation all three components (R,G and B) in the change their values; Figure 40 (a) has R, G and B values of 47, 118 and 64 respectively while Figure 40 (b) has the component values of 90, 226, and 122.

6.3.3 Road Surface Detection

To detect a particular road type (concrete, dusty, asphalt) using the proposed method, we are searching for a set of coefficients that describes precisely a color range. This means that this method is not completely general because it will not work if the road is of the same exact color as the surrounding area such as for instance a dust road in the middle of the desert. However, for the urban and rural environment where the roads are paved with asphalt the color based detection can be quite useful.

The road-detection algorithm is shown in Algorithm 2. The general steps consists in taking the RGB image, convert the image into the HSL color-space (Algorithm 2, line 1). Line 4 computes the *luma* which is an alternative to the intensity that is more perceptually accurate but for all intents and purposes the luma has the same meaning as the intensity or luminosity. Lines 5-7 represent the logical AND operation between each of the HSL layers of the original image with the three masks. These steps represents the filtering of the pixels that do not belong to the specified color range. Line 8 uses the three resulting binary maps and again applies a logical AND between them so that the resulting image represent exactly a binary result where all pixels that are within the color range are white while all filtered pixels are simply black.

Algorithm 2 Object Detection using HSV color-space

0: load image as RGB
1: (h,s,v) = rgb2hsv(rgb)
2: h = h component of hsv
3: s = s component of hsv
4: i = rgb2luma(rgb)
5: $l\_hb = h \wedge (H)$
6: $l\_sb = s \wedge (S)$
7: $l\_ib = i \wedge (I)$
8: result $= l\_hb \wedge l\_sb \wedge l\_ib$

A visualization of the proposed method is shown in Figure 41. The numbered arrows in Figure 41 corresponds to the steps of the algorithm: 0 - represents the loading of the RGB image (line 0), 1 - represents the decomposition to HLS (lines 1 - 4), 2 represents the H, S and L image masks (lines 5 - 7), 3 - corresponds to the creation of the image mask M (line 8) and 4 - corresponds to the blurring of the mask in order to remove noisy pixels (line 9). Observe that lines 5-7 in pseudo-

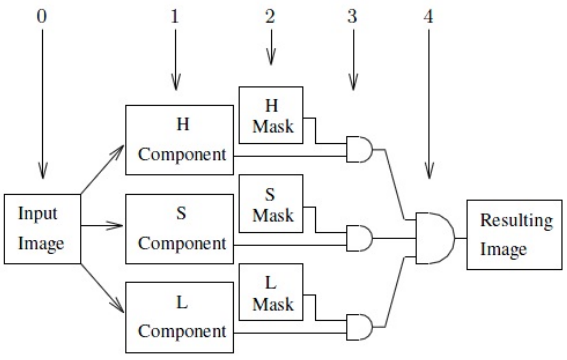


Figure 41: The schematic representation of the proposed color-tracking system.

code 2 represent the three masks by three corresponding sets of values. Let $H =$

$\{h_0, \ldots, h_k\}$, $S = \{s_0, \ldots, s_k\}$ and $I = \{i_o, \ldots, i_k\}$ be the three sets corresponding to the allowed values of Hue, Saturation and Luminance respectively, then any such pixel with h, s and i obeying the condition in eq. 65 is considered to belong to the road surface.

$$\begin{aligned} \exists j |\ & h = h_j \pm \delta \\ & \wedge\ s = s_j \pm \delta \\ & \wedge\ i = i_j \pm \delta \end{aligned} \tag{65}$$

with δ being a constant representing the freedom in mismatching a color to be or not to be part of the road surface. This means that the lines 5-8 in pseudo-code 2 should in fact be represented as a loop shown in pseudo code 3. Observe that within each iteration, three masks are generated and logical AND is applied in between them. The result of this is the that only pixels with exactly the values h_m, s_m, i_m are used to filter the image. At each iteration, the resulting binary image is added to the final result using a logical OR.

Algorithm 3 Loop showing the iterative filtering of the input image

for $m = 1$ to $|H|$ **do**
  5: $l\_hb = h \wedge (h_m)$
  6: $l\_sb = s \wedge (s_m)$
  7: $l\_ib = i \wedge (i_m)$
  result$\vee = l_hb \wedge l_sb \wedge l_ib$
end for

6.3.4 Mask Creation

In pseudo-code 3 each mask contains m values. Initially, a single color was used for detecting the road but such approach is not efficient due to the variations of the road color as described in the introduction. Thus, it is necessary to obtain a range of colors that describe the road without describing the surrounding environment. For this, we create three histograms from prepared road samples representing each component of HSV and select 20 to 30 most used color values that are then used as points representing road in HSV color space. This representation is later done automatically when points are estimated by ANN.

Figure 43 shows the difference when matching a road by an average value of 20 random samples (a) and when using the 20 random samples directly to determine the road surface (b).

In order to maximize the size of the detected road, first a section of a road (containing white stripes) is extracted manually. From this extracted region all bright regions are detected by thresholding resulting in the area of the road color that we are interested in. Example of an sample road and the subsequent white

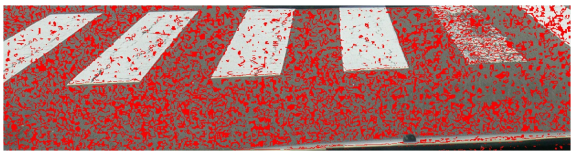


Figure 42: Comparison of road detection: (a) using average values of HSL from 20 samples

Figure 43: Comparison of road detection: (b) using 20 samples to match road directly.

Figure 44: (a) Example of a sample road before lane marking extraction.

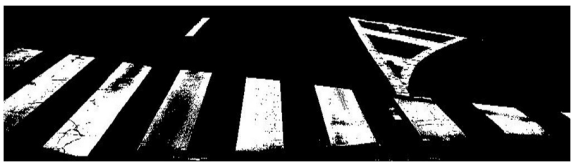


Figure 45: (b) sample road with detected white stripes and markings.

stripes extraction is shown in Figure 44. The remaining area (black in Figure 45) is then sampled for a set of data samples pixels from each component of the HSL color-space. These three data sets of sample pixels are used as points that are considered to be acceptable values representing the color of the road. In such manner a set of example images can be tested on the quality of this manual method. Some results of this method are shown in Figure 47. One of the problems when

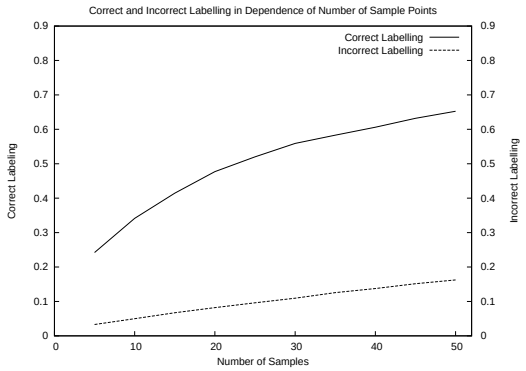


Figure 46: Graph showing the dependency of road detection and road misdetection as a function of number of sampled pixels.

determining the number of sample values in the mask is to determine how many sample values are required. Figure 46 shows statistical average over 95 images of the road detection and the misdetection as function of the number of sample values. As can be seen in the graph (Figure 46) upper line the amount of detected road surface is follows a log-like detection quality. This means that at first there is a big difference between the amount of detected road when using 5 or 10 sample points. However after approximately 30 sample points the difference become lin-

ear. On the other hand, the amount of undesired detected points - pixels located outside of the road surface - grows linearly independently of the sample number. This means that from a practical point of view selecting 30 samples is the best ratio between quality of road detection and background noise detection.

Figure 47: Example of manual HSV-threshold based road surface extraction. The white areas correspond to color matching the sampled color from the road.

6.3.5 Pixel-Density Algorithm

Once the color of the road has been determined and the road has been detected, it is however necessary to remove the noise resulting from matching the road color in the surrounding environment. For this a density-detection algorithm is used. The principle of the density algorithm is very simple: for all white pixels, create super-pixels of size NxN based on the density of the surrounding white pixels. This means, that a road surface that is detected with a constant density of white pixels will be be transformed to a solid white surface while other smaller patches will be eliminated. Example of the result of the density algorithm of roads detected and shown in Figure 47 are shown in Figure 48 The algorithm used for the pixelization is based on the calculation of the average density of white pixels - representing the detected color - and then determining the ratio that

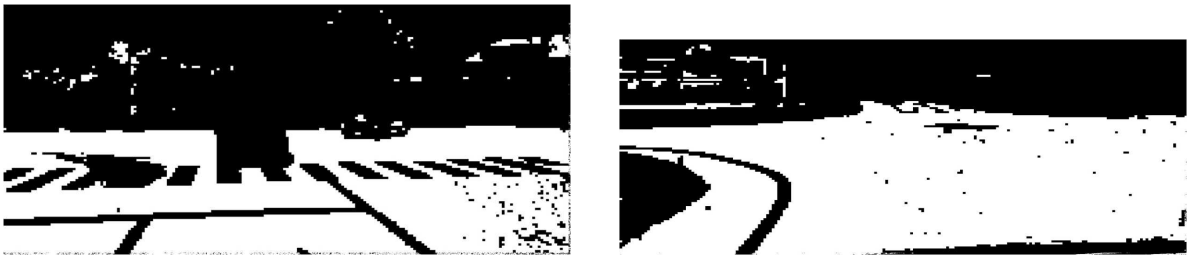


Figure 48: Example of road pixelization based on the relative density of the detected color on input images from

transforms individual pixels to either white or black super-pixels. The equation used to determine the threshold ratio is:

$$p = 0.7 * stepA * stepB * \frac{\sum_{i,j=1}^{i<width,j<height} p(i,j)}{width * height} \tag{66}$$

with $stepA * stepB$ represents the size of the region to pixelize and

$$\sum_{i,j=1}^{i<width,j<height} p(i,j)$$

The final pixelized image is then filtered by relative size of the white blobs to leave only the largest one corresponding to the road surface. This allows us to remove most of the unwanted noise generated detected color patches. Thus it is very important that the road is detected with the highest density of pixels otherwise the interference from the non wanted pixels will make the method very unreliable.

6.3.6 Artificial Neural Networks - MLP

One of the main problems when detecting a road by its color are the variations due to shadow, changing color of the light and humidity. Moreover, when detecting a road in a sequence of images, e.g. a movie the inter image variation provides another level of difficulty to the robust tracking of the road problem. In order to properly determine the set of values representing the color of the tracked road one has to determine the color and the adaptively change the thresholds when required. Our approach is to use Artificial Neural Network (ANN) that will be taught on a set of samples; a set of manually prepared images where the thresholds for detecting road surfaces that are manually extracted and are used as the target of the machine learning. Precisely, the setup uses a Multi-Layered Perceptron (MLP), a feed-forward ANN with the back-propagation learning rule [33, 34]. Figure 49 shows the schematic representation of the ANN used in this chapter. The input to the ANN are two matrices each with k*l coefficients representing the 2D histogram of the combined values of the Hue with Intensity (Luminance)

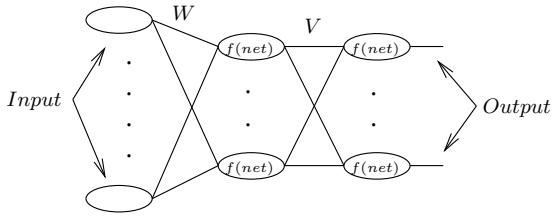


Figure 49: Schematic of an fully connected, feed-forward Artificial Neural Network (ANN).

and Saturation with Intensity. However, such matrices do not represent the coordinates of the averaging from the image, but the represented values are ordered in the natural order of the HSL color map (Figure 39). For instance, the first column of H*L matrix represent points with the hue < h_0, second column points satisfying $h_0 \leq$ hue $< h_1$, etc. Similarly for the rows but with the Intensity as variable. This means that no matter where the road is located on the input image, its representation on the H*S and H*L matrices will not change position. The position change is only triggered by a color change. Such space independent is ideal for ANN as it eliminates any possible problems related to rotations and scaling of the detected object.

The resulting matrices are normalized and then fed to the ANN. Because each of the coefficient in the H*L and S*L matrices are averages of the pixels whose values are in the range $h_j \geq h_{h,y} \geq h_k$ and $l_m \geq l_{x,y} \geq l_n$, the output values of the ANN cannot be more precise than the resolution between of the input matrices. For instance, assume that the desired outputs of the ANN are $O = \{o_0, o_1, o_2, o_3\}$ and their values all fall within a single cell of the H-S and H-L matrices. This means that the road samples (the ANN target outputs) are represented by a single input cell. This means that two different images with the same road values will in fact require a many to one mapping and thus will reduce the learning of the ANN.

However, the discretized matrices H*S and H*L are representing values of the dimensional components scaled to the averaged regions. This scaled approach allows at the same time to reduce the amount of input information to the ANN at the price at a lower detection precision. However, as is shown in the Section 6.3.8 the correct averaging can be experimentally determined and is sufficient for a robust real-time performance.

One possibility how to deal with the low resolution of the input matrices is to allow a non even Hue and Saturation splitting. Because each column and each row is bounded by thresholds, it is possible to generate such H*S and H*L matrices that are finely parsed around the tracked Hue and Saturation and has only a coarse granularity in the rest of the color spectrum. The importance of this representation thus results in an invariant input to the ANN, because the input matrix can always be arranged in such manner that the tracked region will be always at

the same input nodes of the ANN no matter the color of the tracked object.

For the road detection problem, the desired input mapping is such that the road is as spread as possible on the H*S and H*L matrices so as the ANN can learn the road spectrum as precisely as possible. Note that the importance of spreading the road across multiple cells in the input matrices assumes that the road is covering a significant part of the image. Figure 50 shows how the H*S

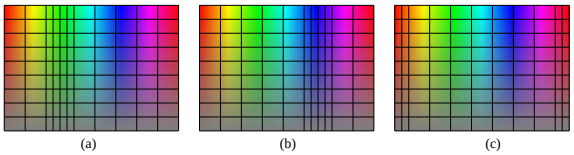


Figure 50: Example of uneven H*S matrix splitting. (a) spitting when the tracked color is green, (b) splitting when tracked color is blue, (c) splitting when tracked color is red

matrix represents the color spectrum given by the Hue and Saturation dimensions for different tracked colors. Observe that as the dimension of the matrix does not change, the same ANN can be used to detect all three colors by simply shifting the inputs of the matrix to the left or to the right. Such ANN needs only to be trained on a single color to learn how to estimate the proper thresholds.

6.3.7 Estimating the Mask Sets

To estimate the values for the H, S and I masks the ANN is trained in a supervised manner. The ANN is taught from a set of examples where the road has been partially extracted and isolated and a set of training values have been obtained. The architecture of the ANN is a classical MLP [33, 34] using the back-propagation rule. The necessity of using the MLP is due to the fact that the HSL mapping is not linear.

The ANN input layer has 800 neurons arranged in a 20×20 H-S (Hue- Saturation) and H-L (Hue-Luminance) matrix representing the values for the various color components as obtained from the H and S component images. The hidden layer has 100 neurons and the output layer has sixty to ninety neurons depending how large the H, S, and I sets are to be.

The activation function for an artificial neuron i is the logistic function given by:

$$f(net_i) = \frac{2}{(1 + exp(-2 * net_i))} - 1 \tag{67}$$

where net_i is given by:

$$net_i = \sum_{p=\alpha}^{\gamma} w_p x_p \tag{68}$$

where the range given by $[\alpha, \gamma]$ represents all neurons connected to the neuron i . The ANN was implemented in Matlab. The learning was performed on $\frac{2}{3}$ of the data samples and the testing was performed over the remaining $\frac{1}{3}$.

6.3.8 Results and Discussion

The training set contained 95 images taken from inside a vehicle during a drive from point A to point B. All 95 images have been manually prepared (the road surface was extracted manually) and 80 of the all available images have been used for the ANN training. The 80 images used for training were randomly selected and the remaining 15 were used for the evaluation of the ANN performance. Several learning runs have been performed and the best have been kept. The result of the ANN was evaluated using the comparison of the detected roads surface vs. the overall noise (F-measure).

To evaluate the performance in a formal manner, we compared the ANN performance with a human road extraction. First the ANN with an even input grid was tested and the results are shown in Figure 69. Figure 69 (a) shows the comparison of the ANN compared to the human manual road extraction before the pixelization algorithm and Figure 69 (b) shows the same results after the pixelization algorithm. (Corresponding figures can be found in appendix A)

Comparing the results from Figure 69 and Figure 70 (a) the improvement of the ANN is directly observable and is as expected given the postulates in Section 6.3.6. The reason of doing such a comparison before and after the pixelization and for various input grid arrangement, is that the F-measure used for the evaluation is deceptive. This means that in many cases, even if the performance of the ANN is poorer than the one performed by the human user, the actual road is detected with higher pixel density and in an more distinguishable manner. this means that in fact there are two criteria when detecting a road by its color:

1. The density of the road detection color vs. the density of the detected noise
2. Discernibility of the road from the rest of the detected objects

Criterion 1 is related to the fact that even if a lot of noise is detected by estimating the color of the road, it can be irrelevant as long as its density is much lower as the detected pixels that belong to the road. This is because once the pixelization algorithm is applied, it can be parameterized so that areas of the image that contain low density of detected pixels will be automatically removed. Criterion 2 represents the fact that the road should be detected as much as possible separated from other detected pixels. This is because in the simplest form the pixelization algorithm, will consider a set of largest blobs as the road. Thus if the road is connected by areas of high density of detected pixels to some other ares such as buildings or trees the resulting blob will be deformed and will not correspond to the geometry of the road.

As an illustration, some of the obtained results are shown in Table 11. As can be seen the ANN was able to learn the presented example situations and was able to confirm the learned information. The rows in the table show the identification number of each test (Id.), the manual result of detection (direct filtering with 20 sample values), and the result obtained after the application of the ANN generated mask values(ANN). The first two rows shows the cases where the ANN's result is worse than the manual extraction, the middle two rows show where the ANN's result is equivalent to the manual extraction and the bottom two rows are examples of cases where the ANN outperformed the manual road extraction. In all images, the white pixels represent the detected pixels.

From Table 11 several details concerning the two criteria introduced above can be observed. For instance, taking the case 2, one can see that even if manual segmentation generate a result with less external noise, it is in fact quite equivalent to the ANN because the pixelization will generate the same amount of noise pixels as the area representing the road is connected by high density areas to non road image elements. Similarly, the case 4, one can observe that the ANN result is better for evaluation - even if using the F-measure it is worse - because the relative density of the detected road is higher than the relative density of the non road areas.

6.4 Road geometry detection using Hough transform

In previous chapter, we introduced a new method of road detection based on its color but there is also another feature common for many roads that can be exploited and make our algorithm more reliable - lane marking. Asphalt roads have often well painted lanes that work as border so we can assume that road is limited by them. Moreover, straight road markings are parallel to each other which means that after their transformation to 2D picture they will meet in point we can consider vanishing point of the road. These premises will help us make road region estimation more reliable without necessity to use complex approaches. The main tool we will use is simple Hough transform for straight lines which has proved to be very useful in many other car detection algorithms

6.4.1 Parralel lines convergence to vanishing point

Since we assume that convergence of parallel lines can create road area border lets make mathematical proof that this will work in all cases. First thing we have to define is how is 3D geometry converted into 2D picture. General transformation of dimensions maps point A with coordinates [X, Y, Z] into 2D space as new point A_t with coordinates $[\frac{fX}{Z}, \frac{fY}{Z}]$. Given this let there be a point A and direction vector

D in three dimensional space. Coordinates of point A will be

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} + \begin{bmatrix} \lambda D_x \\ \lambda D_y \\ \lambda D_z \end{bmatrix}$$

where λ can be any number from $-\infty$ to ∞. To obtain coordinate x we can use formula

$$X = \frac{fX}{Z} = \frac{f(A_x + \lambda D_x)}{A_z + \lambda D_z} \tag{69}$$

if we assume that $\lambda \to \infty$ then direction vector term starts to be dominant and coordinates are obtained like

$$X = \frac{f(\lambda D_x)}{\lambda D_z} = \frac{fD_x}{D_z} \tag{70}$$

it is obvious now that this expression does not depend on A. That means that all parallel lines with the same direction vector D are reaching the same point regardless of their position. Same approach could be used for obtaining Y coordinate and in that case position of vanishing point will be

$$\left[\frac{fD_x}{D_z}, \frac{fD_y}{D_z}\right] \tag{71}$$

for all lines belonging to this family.

6.4.2 Lane detection and geometry estimation

For initial segmentation we used the same algorithm that was used for improving road samples (figures 44 and 45). Resulting image can be seen in 51. After we ob-

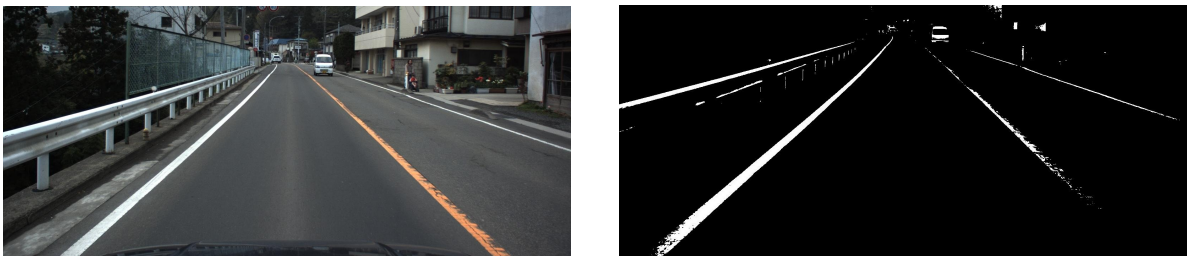


Figure 51: Intial lanes segmentation - original picture and binarized picture.

tained barbarized picture we can use Hough transform to search for straight lines. As a result we will obtain specified number of lines defined by highest amount of Hough peaks in histogram. Result of this can be seen in Figure 52 where resulting line clusters have been highlighted. First step to obtain best lane representation is

to examine these clusters and look for a picture that lies in the middle, all others are discarded afterwards. In the next step we have to get rid of false detections (railing in this case) which can be done by examining the angles of lines. After this we will obtain three lines that correspond to whole road area and middle line separating our driving region from opposite direction. Vanishing point is then lo-

Figure 52: Detected Hough lines in sample picture.

cated in the intersection of leftmost and rightmost detected line. Everything lying outside of the area delimited with lanes is considered background and therefore won't be taken in consideration in final ROI estimation. High level representation of segmented regions is portrayed in Figure 18.

6.5 Final segmentation and car region generation

So far we have described three methods that provide partial results in car detection. Last remaining step to generate our object hypothesis is to connect all together and mark remaining regions as positive detection. Finishing algorithm can be simply described in few steps:

1. Use neural network to obtain road region.
2. If possible find road geometry and create intersection with result from previous step.
3. Detect shadow lines in picture.
4. Discard all shadow lines that do not belong to the region defined in steps 1 and 2
5. Find diameter of remaining shadows and draw rectangles above them in 1:1 ratio.

Practically we can implement this algorithm as logical AND operation between binary pictures (pictures that only contain values 0 or 1) that we got as partial results from previous steps. Problem that we may encounter during this phase is that shadow of the car can be very curvy depending on suns position. To overtake this obstacle we have to save leftmost-lowest position of shadow (this pixel

Figure 53: (a) Original picture of traffic scene

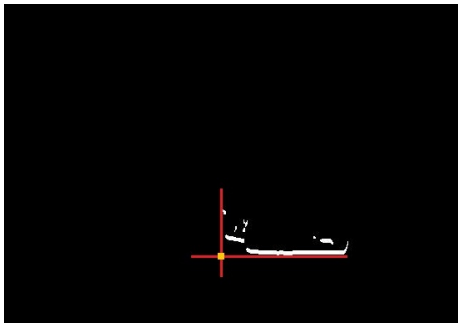


Figure 54: (b) Detected shadow with marked leftmost - bottom point

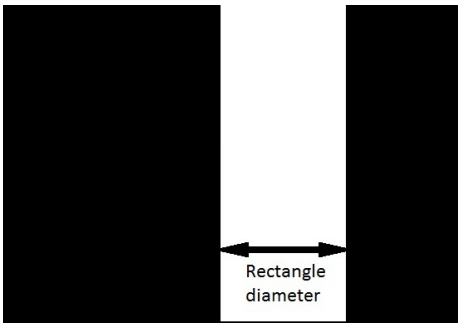


Figure 55: (c) Rectangle side estimation.

Figure 56: (d) Highlighting the resulting ROI.

Figure 57: Visualised ROI generation.

is usually located right next to cars tire, Figure 57 (b)) and mark everything above the shadow as detected area (Figure 57 (c)). Regardless of how curved the detected shadow is we will obtain rectangle that diameter will be used as side of of final square drawn around the car. Position of this square is then determined by saved indices of shadows leftmost-lowest pixel which is identical with bottom-left corner of square. Visualization of this algorithm is portrayed in Figure 57.

7 Used software and implementation details

Choosing the most suitable developing environment and programming language is not a question that can be answered generally. Whole solution and used resources are heavily dependent on what result do we expect. For instance if we want to process streamed video in real time our best choice is to pick some compiled, unmanaged programming language that can provide results in very short time but makes the development of such an application more difficult and prone to errors connected with bad memory management. Since my task was not to provide real time solution but prototype application for generalized car detection in static pictures it wasn't necessary to optimize time complexity or create assembler-compiled solution. Therefore I have originally decided to use C# programming language and Microsoft's own development tool - Visual studio. I have decided to do so because I was familiar with both programming language and development studio and also because fully manged programming language will make development of more complex algorithms easier and also it provides wide .NET API that is prepared for work with image files. Similar functionality could be achieved using Java but in this particular case C# can provide a lot better results because if necessary it still provides possibility to write unmanaged code using the UNSAFE statement. Codes in C# were created using attachments of [6] and Canny edge detector implemented in [38] C# was used to implement first part of algorithm described in 6.2 C# was sufficient to implement the first part of algorithm described in 6.2 but turned out to be not prepared enough for usage of neural networks. If we wanted to continue with development further we would have to either implement whole neural network logic or use some of open source solutions. Due to these development complications we have decided to use Matlab to use development environment which is extremely efficient in work with large matrices and itself has implemented all common image processing algorithms and also contains advanced neural network management tool.

7.1 Development in C#

Since in the beginning I had no previous experiences with image processing I have decided to implement basic operations using tutorials and examples from [6]. While the algorithms themselfs were easy to understand in high level programming language the main problem turned out be automatic memory management. A good example of this problem would be image conversion from full-color to grayscale (which is more suitable for further edge detection etc.). While using the standard .NET library implementing this problem is really simple. We just have to iterate through every pixel of the Bitmap instance which is represented as a 2D array and access pixel at current position which can be also easily obtained through object model of .NET, then compute simple conversion and cre-

ate a new bitmap from new brightness values. This programming approach can be seen in Listing 2. While this approach was quite useful for learning and testing on small images in case of bigger resolutions it is almost impossible to use it because it is extremely slow. Main problem here are the methods GetPixel and SetPixel of the Bitmap object which are always locking the bitmap in memory, then get the information we need and unlock the bitmap again. This operation itself is quite slow the bigger the image is the more times it needs to be used. In our case (images of size 1600x1200) we had to perform it almost 4million times which resulted in processing time up to one minute for a single image. This is when the UNSAFE clause of C# becomes useful. Unsafe modifier can be used for declaration of method, type or a member. After everything enclosed will be considered an unsafe context and requires operations using standard pointer arithmetic. Benefit of this approach in our case is that we only have to lock the bitmap once in the memory. After that we can iterate through whole picture using pointers, compute brightness values and finally unlock the memory again. To do so we need to understand representation of Bitmap in memory. It is encoded as blocks of four bytes representing red, blue and green component and alpha channel of every single pixel. We just have to remember the "beginning point" (Scan0) of the image and then move left in memory. This approach can provide results as much as $30\times$ faster than the previous one and you can find its implementation in Listing 3 (attachment C). Other methods involving Bitmap processing had to be also modified in similar way as unsafe to make testing easier and faster. The final implementation included the whole shadow detection described in [19] using different edge detection algorithms. Results of different edge detection methods were compared only optically because this development branch was later closed.

One of the advantages of .NET framework was simple creation of GUI for testing using Windows forms which can be seen in Figure 63. Visual components of windows forms API allow us to easily pick picture to be processed and then run any of provided algorithms while instantly getting visual result. Comparison of different results while using different edge detection methods can be seen in Figure 62. Object oriented and delegate based GUI also allows anyone (typically supervisor) to instantly use, test and compare results without prior knowledge of given algorithm.

While C# provided good results in shadow detection and segmentation it turned out to be insufficient when it comes to neural networks since .NET itself doesn't contain libraries for ANN computing. After the first idea of using neural networks C# was still used for some initial testing and preparation. For instance for prototype algorithm for manual detection of the road which included conversion to different color space (code example in listig 1). Testing different ways of segmentation such as manual detecting of the road based on average values of hue, segmentation based on multiple values of hue, segmentation using values of RGB or detection of lane markings. Reasons, why to make this test envrionment

was to find a set of features that can successfully distinguish more than 80% of the road. These features were later used as initial data for ANN training. Results obtained from prototype methods can be seen in Figure 43

```
internal static void ConvertRGBToHSI(byte red, byte green, byte blue, out int hue, out int
    saturation, out int intensity )
    {
        saturation = 0;
        hue = 0;
        //INTESNITY
        intensity = (int)((red + green + blue) / 3.0);

        //HUE
        double hueUp = ((red − green) + (red − blue)) / 2.0;
        double hueDown = Math.Sqrt(Math.Pow(red − green, 2) + (red − blue) ∗ (green − blue
            ));
        if (hueDown != 0)
        {
            double h = Math.Acos(hueUp / hueDown);
            if (blue > green)
                h = (2 ∗ Math.PI) − h;
            hue = (int)((180 / Math.PI) ∗ h);
        }

        //SATURATION
        int satDown = red + green + blue;
        if (satDown != 0)
            saturation = (int)((1 − (3.0 / satDown) ∗ Math.Min(Math.Min(red, green), blue)) ∗
                100);
    }
```

Listing 1: Implementation of conversion betwen RGB and HSI colorpsaces in C#.

Windows forms GUI description (Figure 63):

1. Select picture to be processed.
2. Find road sample for threshold estimation using horizontal sobel operator.
3. Find road sample for threshold estimation using vertical sobel operator.
4. Find road sample for threshold estimation using prewitt operator.
5. Find road sample for threshold estimation using canny operator.
6. Mark shadow regions using Canny operator.
7. Mark shadow regions using Prewitt operator.
8. Mark shadow regions using horizontal Sobel operator.
9. Mark shadow regions using vertical Sobel operator.

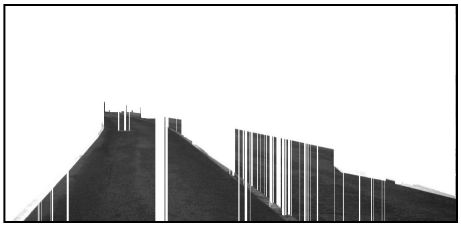


Figure 58: (a) Road sample after using horizontal sobel operator.

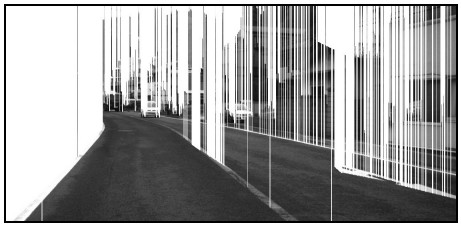


Figure 59: (b) Road sample after using vertical sobel operator.

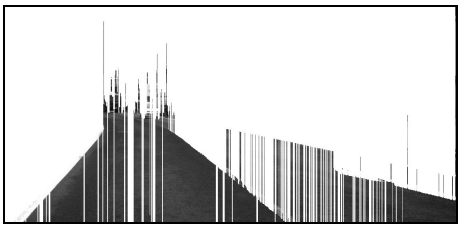


Figure 60: (c) Road sample after using prewitt operator.

Figure 61: (d) Road sample after using canny operator.

Figure 62: Optical comparison of road sample detection.

10. Find edges of shadows shiting image 2px to the top (using canny operator).

7.2 Development in MATLAB

MATLAB (short for matrix laboratory) is interactive programming environment and high level scripting programming language. As our development environment was chosen due to prototype character of our application, where rather than re-implementing state of the art image processing logic we can use prepared libraries and matrix processing logic that is already included. First step of our MATLAB solution was to convert all our previous logic into MATLAB script. The main difference was the representation of our data. While in C# most of are data was represented as objects MATLAB defines all variables (including scalars, matrices, vectors, structures and even objects) as MATLAB array (in C/C++ is declared as mxArray). This difference can be seen for instance in the picture representation. While in C# I had to look at the pictures as one simultaneous piece of memory MATLAB automatically converts them into three dimensional array which can be then simply processed or divided into particular elements of given color space. Example of matlab procedure that is finding 20 most used HSI components is shown in Listing 4 (attachment C).

Simple data processing had a positive impact on hypothesis testing where I was able to compare results of different approaches just by adjusting the indices in matix access operations. This way I was able to make very fast comparison

Figure 63: GUI example

of manual road detection efficiency depending on how many features are used to detect it just by changing parameters in sample code 4 in attachment C. (resulting graph shown in Figure 46). Fast matrix transformations and easy data visualization also allowed us to see possible shortages in data preparation for ANN training. The idea described in Figure 50 can be visualized and examined using MATLABs heatmaps which can show us places where we are loosing information because the segmentation being not fine enough. As we can see on Figure 68 the initial segmentation based on 20×20 matrix was not efficient enough because it was based on an assumption that division of pixels in HSI colorspace is even respectively that it fits the Gaussian distribution. But as we can clearly see distribution is creating peaks and clusters in some places (brightest areas) while other parts of the histogram ale practically empty (black areas). As we can see on first two images in Figure 68 20x20 histogram can differentiate huge differences in distribution but it looses details in same places because it connects empty areas and areas with high density into solid parts. On the 40×40 histogram it is much easier to spot the peaks but it creates unnecessary large amount of data which may lead to over-fitting. These assumptions lead us to an idea of creating an uneven histogram with less data created by clustering empty spaces in 40×40 histogram. That way we obtain 20x20 or 30x30 matrix where the peaks are still differentiated fine enough but empty spaces are clustered into single row to reduce amount of data. A simple implementation of histogram creation can be

seen in attachment 5.After obtainig optimal data input and output the last thing was to create enough data for ANN training. To create input data I have created histogram HS and HV histogram for every picture. These have been merged and transformed into 1 dimensional array creating one row of input data. Similar to that I have taken vectors including most occurring values of HSI components and merged them into single array creating one row of output data. This procedure was done on 95 samples with different input and output types (even histograms, uneven histograms, 60 output values, 90 output values...) and final matrices used for training.

7.2.1 Neural networks in MATLAB

For the actual training I have used MATLABs Neural Network Toolbox which provides supervised learning with feedforward, radial basis, and dynamic networks. It also supports unsupervised learning with self-organizing maps and competitive layers. NNtoolbox provides functionality for creating, training and simulating neural networks for data-fitting (including time-series data), pattern recognition and clustering. It can be controlled through set of command line functions but it also has graphical interface that can be invoked by using nnstart command. I was using the fitting tool which basically works as universal function approximation tool which finds connection between input and output data.
Working with MATLAB neural network toolbox:

1. Choosing fitting tool (attachment D Figure 72)
2. Choosing input and output data (attachment D Figure 73)
3. Dividing samples into training, validation and testing group. (attachment D Figure 74)
4. Adjusting the network architecture (number of hidden neurons) (attachment D Figure 75)
5. Saving the results (attachment D Figure 76)

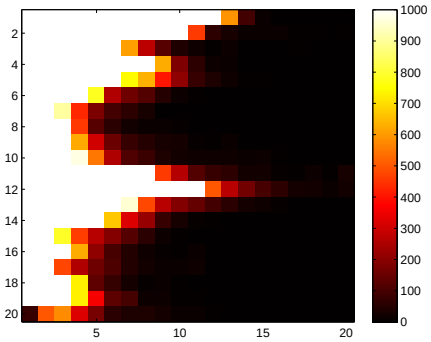


Figure 64: (a) Heatmap of HS histogram with 20x20 resolution.

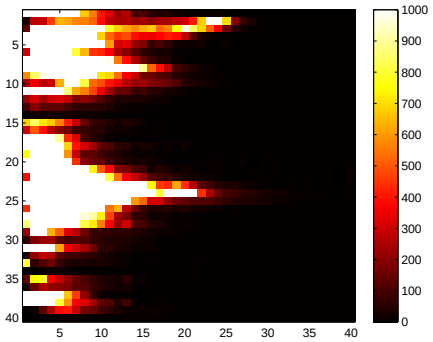


Figure 65: (b) Heatmap of HS histogram with 40x40 resolution.

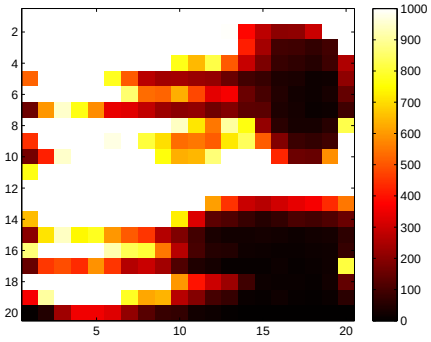


Figure 66: (c) Heatmap of HV histogram with 20x20 resolution.

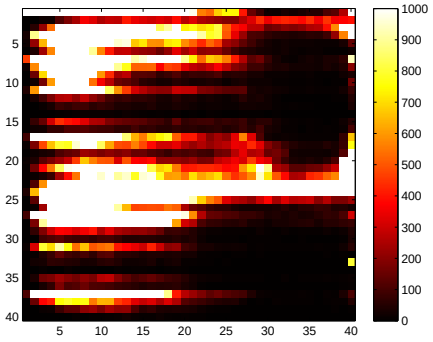


Figure 67: (d) Heatmap of HV histogram with 40x40 resolution.

Figure 68: Graphical output of 2D histogram with different levels of fineness.

8 Conclusion

In this thesis we proposed an automated system for road color detection in real time based on machine learning from manually prepared samples. Unlike in the previous approaches the system is fully autonomous and does not require any additional device beside the camera. The information obtained from the HSL space is sufficient for a successful approach to color detection and can be used for tracking of color blobs.

The future work includes but is not limited to the use the proposed approach with different neural network structure and parameters as well as verification of the detection system for a multi-color detecting system. This initial work is also a starting point for developing an adaptive robotic platform based on the concepts of Adaptive functional Module Selection (AFMS) proposed in [35, 36]. Following this approach it is possible to mix various algorithms adaptively based on environmental conditions and thus not only to save resources but also to provide a satisfactory result of processing at a lowest possible computational cost. Moreover, from the results of the learning of the ANN, in some cases the networks that were generated are detecting very well in particular conditions while doing very poorly in others. This means that having a set of such network for different type of input images, one can efficiently select the best detector on a case by case basis.

Along with new method proposal we put it into use in combination with classical car detection approaches in order to create robust and reliable solution for complex car detection. Final algorithm has been tested on a set of pictures created with cameras mounted in the front part of vehicle. Pictures were created under various weather conditions. Algorithm was also tested on pictures with additional noise and proved to be very stable even in very bad conditions.

Certain parts of this theses have already been published in the paper [37] (mainly chapter 6.3) that was written along with dr. Martin Lukáč and help of professor Kameyama.

Slavomír Truchlík

9 References

[1] W. Jones *Keeping Cars from Crashing*,IEEE Spectrum, vol. 38, no 9, pp. 40-45, 2001.

[2] W. Jones *Building Safer Cars*,IEEE Spectrum, vol. 39, no. 1, pp. 82- 85, 2002.

[3] Robert A. Maschal Jr., S. Susan Young, Joe Reynolds, Keith Krapels, Jonathan Fanning, and Ted Corbin *Review of Bayer Pattern Color Filter Array (CFA) Demosaicing with New Quality Assessment Algorithms*,Technical report, U.S. Army Research Laboratory, 2010.

[4] Cok, D. R. *Signal Processing Method and Apparatus for Sampled Image Signals.*, U.S. Patent 4 630 307, December 1986.

[5] Darrin Cardani *Adventures in HSV Space*

[6] Michal Dobeš *Zpracování obrazu a algoritmy v C#* Nakladatelství BEN - technická literatura, Praha 2008 [in Czech]

[7] Ramanath, R.; Snyder, W. E.; Bilbro, G. L.; Sander III, W. A. *Demosaicking Methods for Bayer Color Arrays.* Journal of Electronic Imaging July 2002, 11 (3), 633–642.

[8] Eduard Sojka *Digitální zpracování a analýza obrazu* ISBN 80-7078-746-5, 1999 [in Czech]

[9] D. Maar, E. Hildreth *Theory of edge detection* Proc. Roy. Soc. London B207, 1980 187-217

[10] Henrique S. Malvar, Li wei He, and Ross Cutler. *High-quality linear interpolation for demosaicing of bayer-patterned color images.* In Proceedings of the IEEE International Conference on Speech, Acoustics, and Signal Processing, 2004.

[11] John Francis Canny *Finding Edges and Lines in Images* Technical report 720, MIT Artificial Intelligence Laboratory

[12] John F. Canny *A Computational Approach to Edge Detection* IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. PAMI-8, NO. 6, NOVEMBER 1986

[13] Jeff Heaton *Introduction to Neural Networks for C#, 2nd Edition* Heaton Research Inc., ISBN 1-60439-009-3, 2008

[14] Peter Sinčák, Gabriela Andrejková *Neurónové siete - Inžiniersky prístup* Katedra kybernetiky a umelej inteligencie, Elektrotechnická fakulta, Technická Univerzita Košice, 1996 [in Czech]

[15] Andrew Ng *Výukové materiály Machine Learning* Available online na <https://www.coursera.org>, 2012

[16] Ivo Vondrák *Neuronové sítě* Fakulta elektrotechniky a informatiky VŠB - Technická univerzita Ostrava, katedra informatiky, 2009 [in Czech]

[17] Rosenblatt F. *Principles of Neurodynamics Perceptron and the Theory of Brain Mechanism* Spartan Press, Washington, 1962

[18] Zehang Sun, George Bebis, Ronald Miller *On-Road Vehicle Detection: A Review* IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 28, NO. 5, MAY 2006

[19] Christos Tzomakas, Werner von Seelen *Vehicle Detection in Traffic Scenes Using Shadows* Ruhr-Universität Bochum, Istitut für Neuroinformatik, Internal Report 98-06, 1998

[20] H. Mori abd N. M. Carakari. *Shadow and Rythm as Sign Patterns of Obstacle detection* In International Symposium on Industrial Electronics, pages 271-277, 1993

[21] Fu, K.; Gonzales, R. & Lee, C. Freeman *Robotics Control, Sensing, Vision and Intelligence.* McGraw-Hill, 1987

[22] Blake, A. and Yuille, A., *Active Vision* MIT Press, 1992

[23] William K. Pratt *Digital image processing 4th edition* Pixelsoft Inc., Los Altos, California, 2007

[24] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins *Digital Image Processing Using MATLAB* Pearson Prentice Hal, December 26, 2003

[25] Sattar, J. and Dudek, G. *On the performance of color tracking algorithms for underwater robots under varying lighting and visibility.* In Proceedings of the IEEE International Conference on Robotics and Automation. 2006

[26] Hayashi, Y. and Fujiyoshi, H. *Mean-shift-based color tracking in illuminance change.* In Visser, U., Ribeiro, F., Ohashi, T., and Dellaert, F., editors, RoboCup 2007: Robot Soccer World Cup XI, pages 302–311. Springer-Verlag, Berlin, Heidelberg. 2008

[27] Stern, H. and Efros, B. *Adaptive color space switching for tracking under varying illumination.* Image and Vision Computing, 23(3):353–364. 2005

[28] Hariyama, M., Hanyu, T., and Kameyama, M. *A collision detection multiprocessor for intelligent vehicles using a high-density cam.* In Proceedings of the Intelligent Vehicles Symposium. 1994

[29] Conrad, P. and Foedisch, M. *Performance evaluation of color based road detection using neural nets and support vector machines.* In Proceedings of the 32nd Applied Imagery Pattern Recognition Workshop. 2003

[30] Chiu, K.-Y. and Lin, S.-F. *Lane detection using color-based segmentation.* In Proceedings of the IEEE Intelligent Vehicles Symposium. 2005

[31] Huang, J., Kong, B., Li, B., and Zheng, F. *A new method of unstructured road detection based on hsv color space and road features.* In Proceedings of the International Conference on Information Acquisition. 2007

[32] Kong, H., Audibert, J.-Y., and Ponce, J. *General road detection from a single image.* IEEE Trasactions on Image Processing, 19(8):2211–2220. 2010

[33] Rosenblatt, F. *Perceptron: a probablistic model for information storage and retrieval in the brain.* Psycho-logical Review, 65:386–408. 1958

[34] Rumelhart, D. and McClelland, J. *Parallel Dis-tributed Processing.* MIT Press (reprint). 1987

[35] Lukac, M., Kameyama, M., and Perkowski, M. *Adaptive selection of intelligent processing modules and its applications,.* In The 2010 International Con-ference on Artificial Intelligence, WORLDCOMP'10. 2010

[36] Lukac, M., Kameyama, M., and Perkowski, M. *Emotion-aware probabilistic robotics.* In Second In-ternational symposium on Aware Robotics, ISAC2010. 2010

[37] Lukac, M., Kameyama, M., and Truchlik, S. *Learning to detect road colors under a dynamically changing ligt source.* Graduate School of Information Sciences, Tohoku University, Sendai, Japan, Technical University of Ostrava, Ostrava, Czech Republic. 2012

[38] Web page of the AForge.NET framework *AForge.NET* <http://www.aforgenet.com//>.

[39] Figure location on wikipedia page. *RGB cube* <http://en.wikipedia.org/wiki/File:RGB_Cube_Show_lowgamma_cutout_b.png//>.

[40] Figure location on wikipedia page. *Biological neuron* <http://en.wikipedia.org/wiki/File:Neuron_Hand-tuned.svg//>.

A Graphs and comparisments

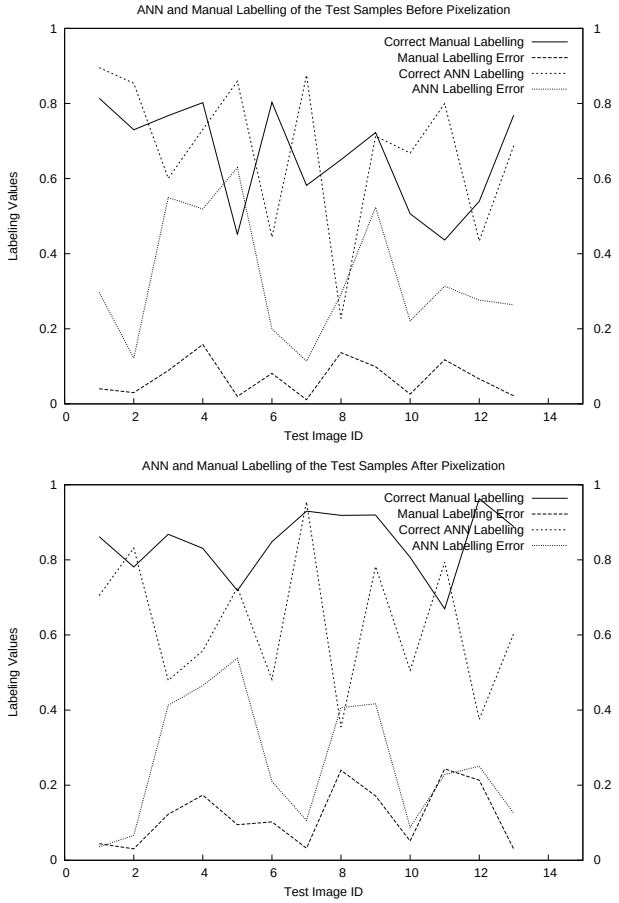


Figure 69: Graphical comparison of the road detection between the learned ANN and a manual detection.

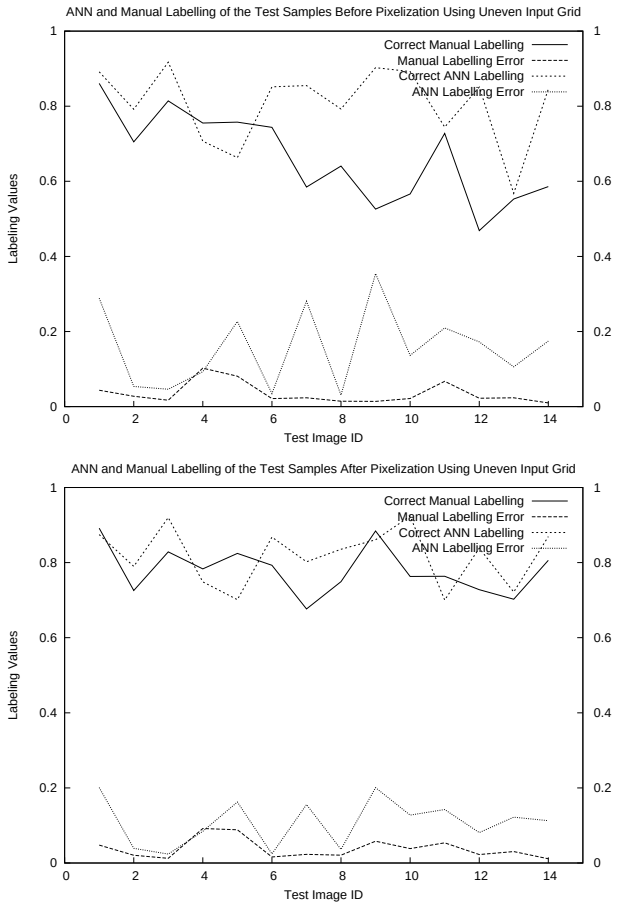


Figure 70: Graphical comparison of the road detection between the learned ANN and a manual detection when the ANN is using an uneven input data as illustrated in Figure 50.

B Results of road detection

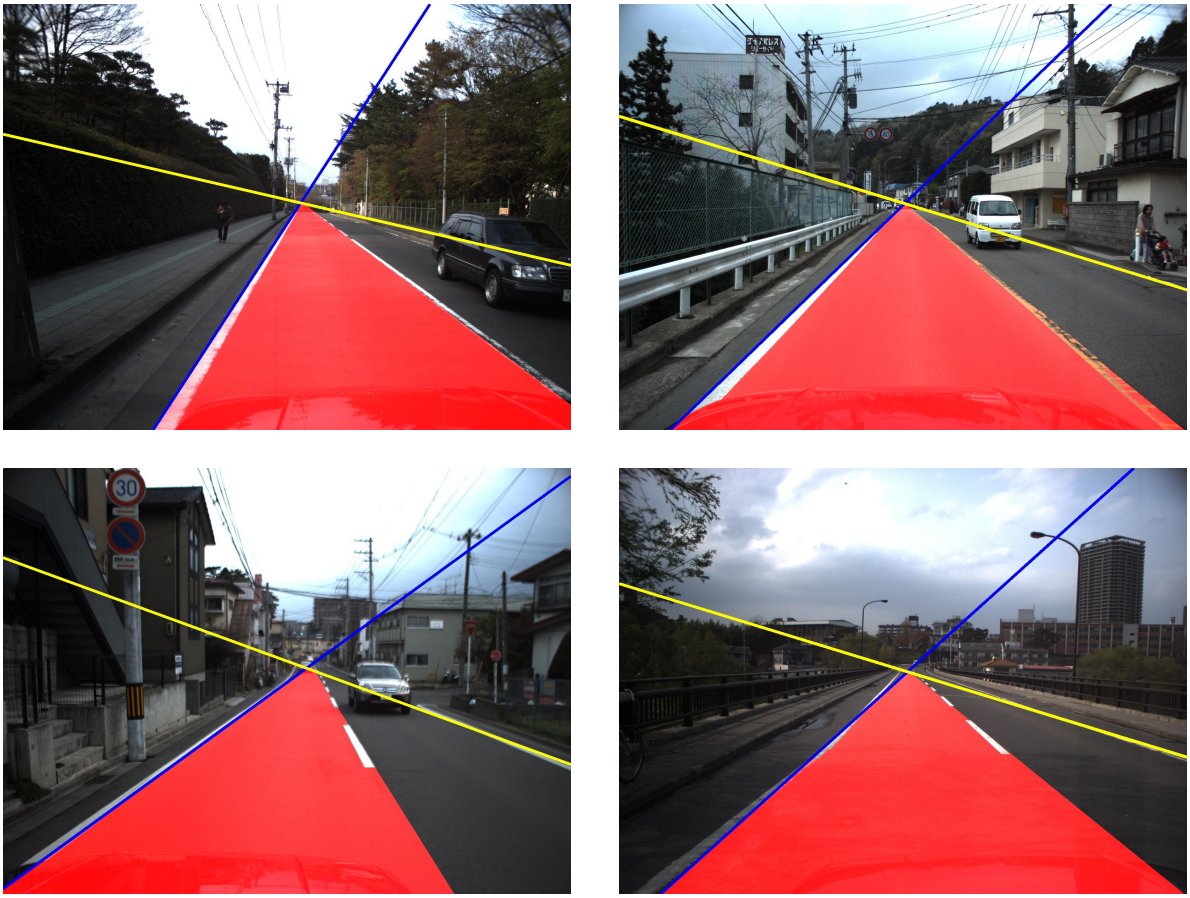
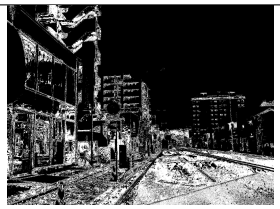
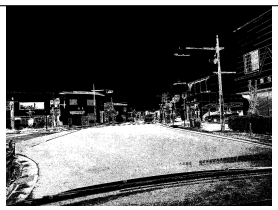
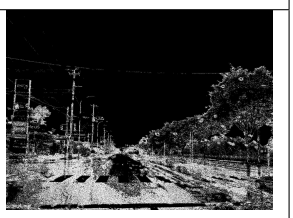


Figure 71: The results of geometry based road surface detection.

Table 11: The results of ANN based road surface detection.

Id.	1	2	3	4
Manual				
ANN				
Id.	5	6	7	8
Manual				
ANN				
Id.	9	10	11	12
Manual				
ANN				

C Code snippets and examples

```
public Bitmap ConvertToGreyScale(Bitmap bMap)
{
  Bitmap returnBMap = new Bitmap(bMap.Width, bMap.Height)
   for (int i = 0; i < bMap.Width; i++)
   {
      for (int j = 0; j < bMap.Height; j++)
      {
            //accessing pixel at the i,j indexes
             Color originalColor = original.GetPixel(i, j);

             //computing the value of brightness
             int greyValue = (int)((originalColor.R * 0.3) + (originalColor.G * 0.59) + (
                 originalColor.B * 0.11));

             //setting the new pixel of greyscale image
             Color newColor = Color.FromArgb(greyValue, greyValue, greyValue);
             returnBMap.SetPixel(i, j, newColor);
         }
    }
   return returnBMap;
}
```

Listing 2: Simple image conversion to greyscale

```
internal static byte[,] ConvertImageToGrey(Bitmap fullcolorImage)
        {
            unsafe
            {
                byte[,] greyScaleImage = new byte[fullcolorImage.Width, fullcolorImage.Height];
                BitmapData originalData = fullcolorImage.LockBits(
                    new Rectangle(0, 0, fullcolorImage.Width, fullcolorImage.Height),
                    ImageLockMode.ReadOnly, PixelFormat.Format24bppRgb);

                for (int y = 0; y < fullcolorImage.Height; y++)
                {
                    byte* oRow = (byte*)originalData.Scan0 + (y * originalData.Stride);

                    for (int x = 0; x < fullcolorImage.Width; x++)
                    {
                        byte blue = *oRow; oRow++;
                        byte green = *oRow; oRow++;
                        byte red = *oRow; oRow++;
                        byte greyScale = (byte)(red * 0.299 + green * 0.587 + blue * 0.114);
                        greyScaleImage[x, y] = greyScale;
                    }
                }

                fullcolorImage.UnlockBits(originalData);
                return greyScaleImage;
            }
        }
```

Listing 3: Image conversion to grayscale using pointer arithmetics.

```
I = imread('D:\diplomka obr\kameyama\roadsamples\1.jpg');

greyScale = rgb2gray(I);

%creating a logical array that discards white spaces in the road samples
blackAndWhite = im2bw(greyScale, 0.9);

%converting image from RGB to HSV colorspace
HSV = rgb2hsv(I);

%rounding values of HSI on 2 decimal spaces
HSV = HSV * 1000;
HSV = round(HSV);
HSV = HSV / 1000;

%dividing 3 dimensional array into components of HSV, all values in matrix are normalized
H = HSV(:,:,1);
%H = H * 360;
S = HSV(:,:,2);
%S = S * 100;
V = HSV(:,:,3);
%V = V*100;

%creting new arrays that only contain pixel that will be processed
H = H(blackAndWhite);
S = S(blackAndWhite);
V = V(blackAndWhite);

%finding most used values
[Value, Index] = unique(sort(H(:)));
M = sortrows([Value, diff([0; Index])], -2);
Top20H = M(1:20, :);
Top20H = Top20H(:,1);

[Value, Index] = unique(sort(S(:)));
M = sortrows([Value, diff([0; Index])], -2);
Top20S = M(1:20, :);
Top20S = Top20S(:,1);

[Value, Index] = unique(sort(V(:)));
M = sortrows([Value, diff([0; Index])], -2);
Top20V = M(1:20, :);
Top20V
Top20V = Top20V(:,1);
```

Listing 4: Finding most used values of HSI components

```
I = imread(path);
hsv_img = rgb2hsv(I);
H = hsv_img(:,:,1);
```

```
H = H * 360;
S = hsv_img(:,:,2) ;
S = S * 100;
V = hsv_img(:,:,3) ;
V = V*100;

HS = zeros(20,20);
HV = zeros(20,20);

HVuneven = zeros(40,40);
HSuneven = zeros(40,40);

for ind = 1:numel(H)
    histValueH = round(H(ind)/18);
    histValueS = round(S(ind)/5);
    histValueV = round(V(ind)/5);

    if histValueH == 0
        histValueH = histValueH+1;
    end
    if histValueS == 0
        histValueS = histValueS+1;
    end
    if histValueV == 0
        histValueV = histValueV+1;
    end
    HS(histValueH, histValueS) = HS(histValueH, histValueS)+1;
    HV(histValueH, histValueV) = HV(histValueH, histValueV)+1;

    histValueHUneven = round(H(ind)/9);
    histValueSUneven = round(S(ind)/2.5);
    histValueVUneven = round(V(ind)/2.5);
    if histValueHUneven == 0
        histValueHUneven = histValueHUneven+1;
    end
    if histValueSUneven == 0
        histValueSUneven = histValueSUneven+1;
    end
    if histValueVUneven == 0
        histValueVUneven = histValueVUneven+1;
    end
     %{
     if histValueHUneven == 0
        histValueHUneven = histValueHUneven+1;
     end
     if histValueHUneven == 15 || histValueHUneven == 16
        histValueHUneven = 15;
     end
     if histValueHUneven == 17 || histValueHUneven == 18
        histValueHUneven = 16;
     end
     if histValueHUneven == 18 || histValueHUneven == 19
        histValueHUneven = 17;
     end
```

```
    if histValueHUneven == 20 || histValueHUneven == 21
        histValueHUneven = 18;
    end
     if histValueHUneven == 22 || histValueHUneven == 23
        histValueHUneven = 19;
     end
    if histValueHUneven > 23
        histValueHUneven = 20;
    end
    %}
    HSuneven(histValueHUneven, histValueSUneven) = HSuneven(histValueHUneven,
          histValueSUneven)+1;
    HVuneven(histValueHUneven, histValueVUneven) = HVuneven(histValueHUneven,
          histValueVUneven)+1;

end
```

Listing 5: Creating HS and HV histograms

```
% This script assumes these variables are defined:
%
%   trainData - input data.
%   resultData - target data.

inputs = trainData';
targets = resultData';

% Create a Fitting Network
hiddenLayerSize = 150;
net = fitnet(hiddenLayerSize);

% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.inputs{1}.processFcns = {'removeconstantrows','mapminmax'};
net.outputs{2}.processFcns = {'removeconstantrows','mapminmax'};


% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivide
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% For help on training function 'trainlm' type: help trainlm
% For a list of all training functions type: help nntrain
net.trainFcn = 'trainlm';  % Levenberg-Marquardt

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'mse';  % Mean squared error

% Choose Plot Functions
```

```
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform', 'plottrainstate', 'ploterrhist', ...
  'plotregression', 'plotfit'};


% Train the Network
[net, tr] = train(net,inputs,targets);

% Test the Network
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs)

% Recalculate Training, Validation and Test Performance
trainTargets = targets .* tr.trainMask{1};
valTargets = targets .* tr.valMask{1};
testTargets = targets .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,outputs)
valPerformance = perform(net,valTargets,outputs)
testPerformance = perform(net,testTargets,outputs)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, plotfit(net,inputs,targets)
%figure, plotregression(targets,outputs)
%figure, ploterrhist(errors)
```

Listing 6: Preparation of neural network

D MATLAB neural network toolbox

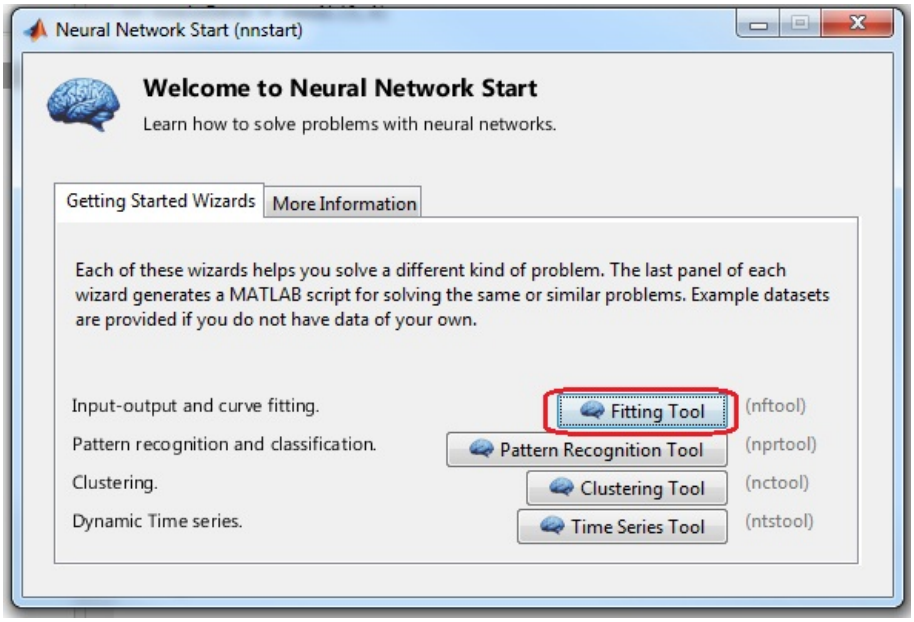


Figure 72: Neural network toolbox

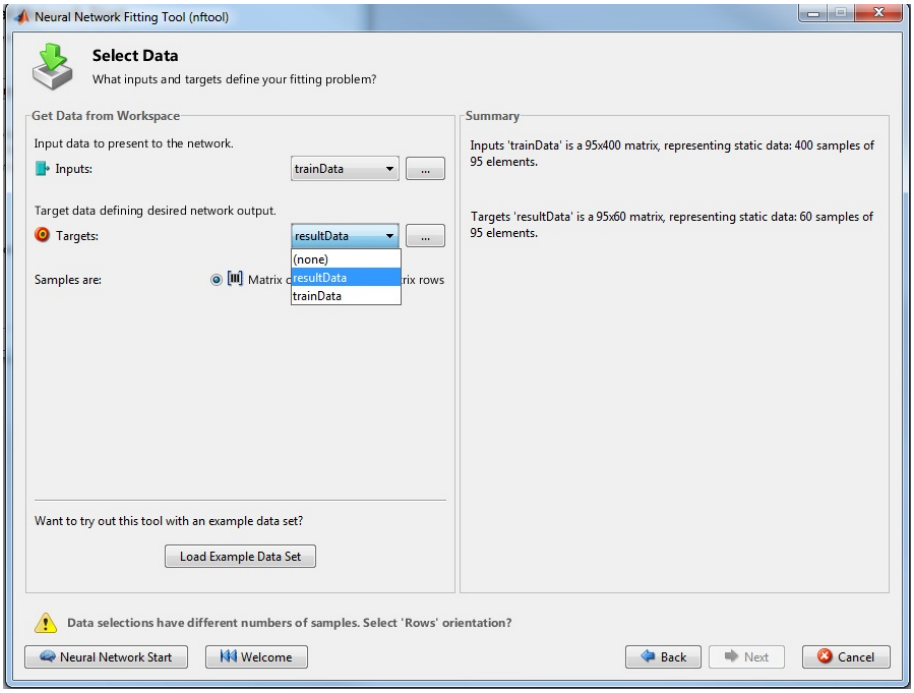


Figure 73: Choosing the input and output set.

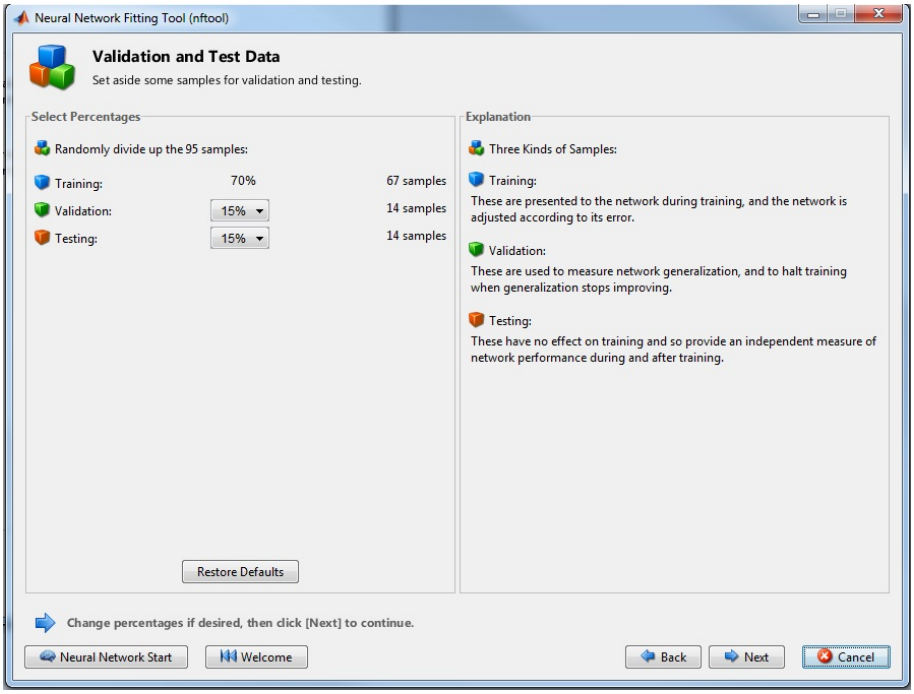


Figure 74: Dividing training set into sub-sets for training, validation and testing.

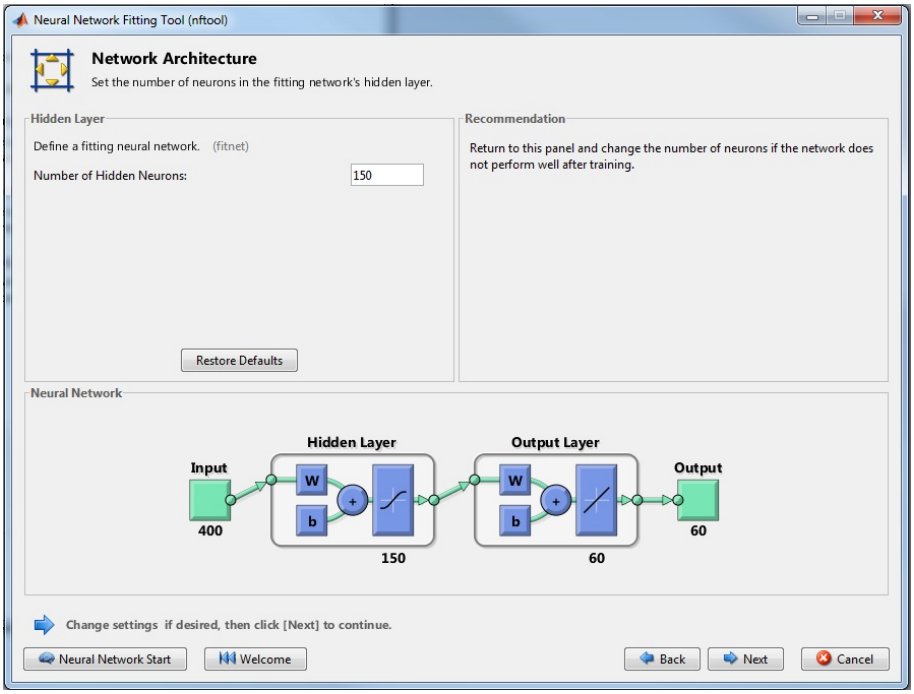


Figure 75: Adjusting the network architecture.

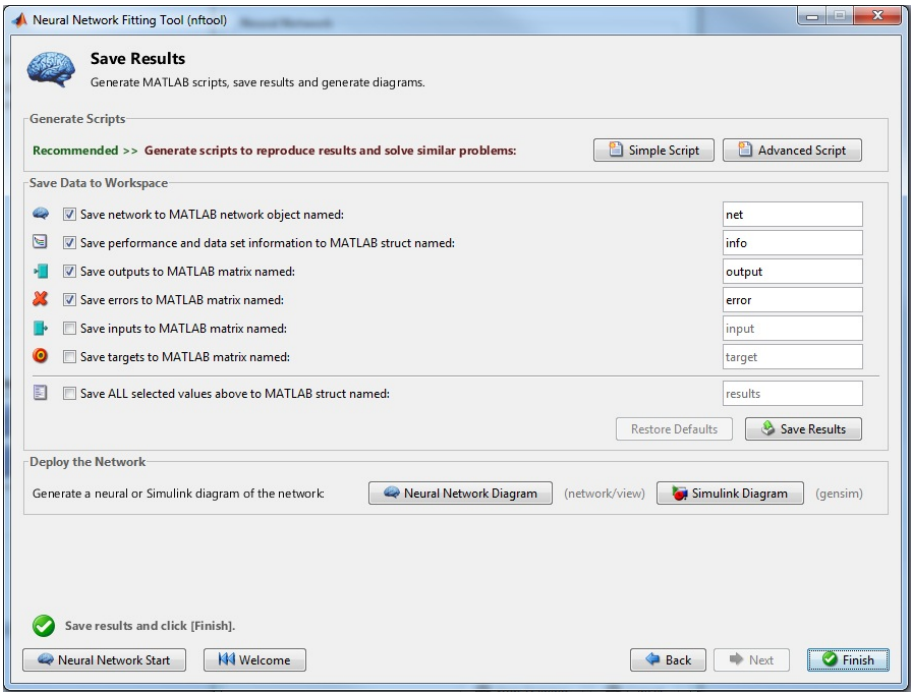


Figure 76: Saving the results.